

TUGBOAT

Volume 22, Number 4 / December 2001

	259	Addresses
General Delivery	261	From the Board of Directors / <i>Arthur Ogawa</i>
	263	Editorial comments / <i>Barbara Beeton</i> The status of <i>TUGboat</i> ; Glitches in <i>TUGboat</i> 22:1/2; Zapfest exhibition, honoring Hermann Zapf; Mordecai Richler font; References for <i>TeX</i> and Friends; Institut d'Histoire du Livre; More historic books online at the British Library; Web document analysis; The little tugboat that could
Electronic Documents	265	<i>execJS</i> : A new technique for introducing discardable JavaScript into a PDF file from a <i>LATEX</i> source / <i>D. P. Story</i>
	269	<i>LATEX</i> , SVG, fonts / <i>Michel Goossens</i>
	280	<i>mimeTeX</i> announcement / <i>John Forkosh</i>
Font Forum	281	Making outline fonts from bitmap images / <i>Karl Berry</i>
Software & Tools	285	Size reduction of chemical structural formulas in <i>X^YMT_EX</i> (Version 3.00) / <i>Shinsaku Fujita</i> and <i>Nobuya Tanaka</i>
	290	The package <i>ps4pdf</i> : from PostScript to PDF / <i>Rolf Nieprasch</i> and <i>Herbert Voß</i>
	292	Instant Preview and the <i>TeX</i> daemon / <i>Jonathan Fine</i>
Graphics Applications	298	Space geometry with <i>METAPOST</i> / <i>Denis Roegel</i>
	314	The plot functions of <i>pst-plot</i> / <i>Jana Voß</i> and <i>Herbert Voß</i>
	319	Three dimensional plots with <i>pst-3dplot</i> / <i>Herbert Voß</i>
	330	Axis alignment in <i>X_Y-pic</i> diagrams / <i>Alexander R. Perlis</i>
	334	<i>Eukleides</i> : A geometry drawing language / <i>Christian Obrecht</i>
Book Review	338	<i>TeX Reference Manual</i> , by David Bausum / <i>Stephen Moye</i>
Hints & Tricks	339	Glisterings / <i>Peter Wilson</i>
	341	The treasure chest / <i>William Adams</i>
	349	Highlighting in the <i>LATEX</i> picture environment / <i>David M. Tulett</i>
Macros	350	A complement to <i>\smash</i> , <i>\llap</i> , and <i>\rlap</i> / <i>Alexander R. Perlis</i>
LATEX	353	Typesetting critical editions of poetry / <i>John Burt</i>
	361	CV formatting with <i>Cv_{tex}Ve</i> / <i>Didier Verna</i>
Abstracts	365	<i>Les Cahiers GUTenberg</i> , Contents of double issue 39/40 (May 2001)
News & Announcements	368	TUG '2002 Announcement
	369	Calendar
	371	TUG '2003 Announcement
	372	Euro _{TeX} '2003 — The 14 th European <i>TeX</i> Conference
Cartoon	260	Ya can't touch us! / <i>Roy Preston</i>
TUG Business	373	Institutional members
	374	TUG membership application
Advertisements	375	<i>TeX</i> consulting and production services
	376	Just Published: <i>TeX Reference Manual</i> by David Bausum
	cover 3	Blue Sky Research

T_EX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published quarterly by the T_EX Users Group, 1466 NW Naito Parkway, Suite 3141, Portland, OR 97209-2820, U.S.A.

2001 dues for individual members are as follows:

- Ordinary members: \$75.
- Students: \$45.

Membership in the T_EX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed. 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.

Periodical-class postage paid at Portland, OR, and additional mailing offices. Postmaster: Send address changes to *TUGboat*, T_EX Users Group, 1466 NW Naito Parkway, Suite 3141, Portland, OR 97209-2820, U.S.A.

Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both T_EX and the T_EX Users Group. For further information, contact the TUG office (office@tug.org).

T_EX is a trademark of the American Mathematical Society.

TUGboat © Copyright 2001, T_EX Users Group

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

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

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

Printed in U.S.A.

Board of Directors

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

Mimi Jett, *President*^{*+}

Kristoffer Rose^{*+}, *Vice President*

Don DeLand^{*+}, *Treasurer*

Arthur Ogawa^{*+}, *Secretary*

Barbara Beeton

Karl Berry

Kaja Christiansen

Susan DeMeritt

Stephanie Hogue

Judy Johnson⁺

Ross Moore

Cheryl Ponchin

Petr Sojka

Philip Taylor

Raymond Goucher, *Founding Executive Director*[†]

Hermann Zapf, *Wizard of Fonts*[†]

^{*}member of executive committee

⁺member of business committee

[†]honorary

Addresses

General correspondence,
payments, etc.

T_EX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Delivery services,
parcels, visitors

T_EX 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
T_EX users:
support@tug.org

To 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 to board@tug.org

[printing date: July 2003]

One of the great ironies of the information age is that, while the late twentieth century will undoubtedly have recorded more data than any other period in history, it will also almost certainly have lost more information than any previous era.

Alexander Stille
The Future of the Past (2002)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP

EDITOR BARBARA BEETON

VOLUME 22, NUMBER 4

PORTLAND

•

OREGON

DECEMBER 2001

U.S.A.

TUGboat

For the 2002 membership year, the communications of the *TeX* Users Group will be published as one double issue and two regular issues. The first issue (Vol. 23, No. 1) contains the Proceedings of the TUG 2002 Annual Meeting.

For 2003, three issues will be published. The first issue (Vol. 24, No. 1) is expected to contain the Proceedings of EuroTeX 2003, and the second issue, the Proceedings of the 2003 TUG Annual Meeting. The third issue will be a regular issue.

We are unfortunately not able to set a definitive schedule for the appearance of the next few issues.

TUGboat is distributed as a benefit of membership to all members.

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

Submitting Items for Publication

Owing to the lateness of the present issue, and the scarcity of material submitted for future issues, items will be processed as received.

Manuscripts may be submitted to a member of the *TUGboat* Editorial Board, as listed at the top of this page (see addresses on p. 259). Articles of general interest or those not covered by any of the editorial departments listed should be sent by electronic mail to

TUGboat@tug.org

All items submitted on magnetic media or as camera-ready copy should be addressed to the Editor-in-Chief, Barbara Beeton, to the Managing Editor, Robin Laakso, or to the Production Manager, Mimi Burbank, with an e-mail notice to the *TUGboat* address.

The *TUGboat* "style files", for use with either plain *TeX* or *LATEX*, are available from CTAN and are on the *TeX* Live CD. For authors who have no network access (browser or FTP), they will be sent on request; please specify which is preferred. Send e-mail to the *TUGboat* address above, or write or call the TUG office.

Reviewers

Additional reviewers are needed, to assist in checking new articles for completeness, accuracy, and presentation. Volunteers are invited to submit their names and interests for consideration; write to *TUGboat@tug.org* or to the Editor, Barbara Beeton (see address on p. 259).

TUGboat Editorial Board

Barbara Beeton, *Editor-in-Chief*

Robin Laakso, *Managing Editor*

Mimi Burbank, *Production Manager*

Victor Eijkhout, *Associate Editor, Macros*

Jeremy Gibbons, *Associate Editor,*

"Hey — it works!"

Alan Hoenig, *Associate Editor, Fonts*

Christina Thiele, *Associate Editor,*

Topics in the Humanities

Production Team:

Barbara Beeton, Mimi Burbank (Manager), Karl Berry, Robin Fairbairns, Michael Sofka, Christina Thiele

See page 259 for addresses.

Other TUG Publications

TUG is interested in considering additional manuscripts for publication. These might include manuals, instructional materials, documentation, or works on any other topic that might be useful to the *TeX* community in general. Provision can be made for including macro packages or software in computer-readable form. If you have any such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee at *tug-pub@tug.org* or in care of the TUG office.

TUGboat Advertising

For information about advertising rates or publication schedules, write or call the TUG office.

Trademarks

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

METAFONT is a trademark of Addison-Wesley Inc.

PostScript is a trademark of Adobe Systems, Inc.

TeX and *AMS-TeX* are trademarks of the American Mathematical Society.

UNIX is a registered trademark of X/Open Co. Ltd.

Addresses

TEX USERS GROUP OFFICE

Robin Laakso
1466 NW Naito Parkway, Suite 3141
Portland, OR 97209-2820, USA
+1 503-223-9994
Fax: +1 503-223-3960
office@tug.org

WILLIAM ADAMS

75 Utley Drive, Ste. 110
Mechanicsburg, PA 17055, USA
willadams@aol.com

BARBARA BEETON

American Mathematical Society
P. O. Box 6248
Providence, RI 02940, USA
+1 401 455-4014
bnb@ams.org, tugboat@tug.org

KARL BERRY

685 Larry Ave. N
Keizer, OR 97303, USA
karl@tug.org

MIMI R. BURBANK

CSIT, 408 Dirac Science Library
Florida State University
Tallahassee, FL 32306-4130, USA
+1 850 644-2440
mimi@csit.fsu.edu

JOHN BURT

Department of English MS023
Brandeis University
Waltham, MA 02454, USA
burt@brandeis.edu

KAJA CHRISTIANSEN

Dept. of Computer Science
Aarhus Univ., Ny Munkegade
Bldg 540
DK-8000 Aarhus C, Denmark
kaja@daimi.aau.dk

DONALD DELAND

Integre Technical Publishing Co.
4015 Carlisle NE, Suite A
Albuquerque, NM 87107, USA
don.deLand@tug.org

SUSAN DEMERITT

IDA/CCR La Jolla
4320 Westerra Court
San Diego, CA 92121, USA
+1 619 622-5455
sue@ccrwest.org

VICTOR EIJKHOUT
Computer Science Department
111 Ayres Hall
University of Tennessee
Knoxville, TN 37996-1301, USA
victor@eijkhout.net

ROBIN FAIRBAIRNS
32 Lilac Court
Cherryhinton Rd.
Cambridge, CB1 4AY, UK
Robin.Fairbairns@cl.cam.ac.uk

JONATHAN FINE
203 Coldhams Lane
Cambridge, CB1 3HY, UK
jfine@activetex.org

PETER FLYNN
Computer Centre
University College
Cork, Ireland
+353 21 902609
pf@ucc.ie

JOHN FORKOSH
285 Stegman Parkway #309
Jersey City, NJ 07305, USA
john@forkosh.com

SHINSAKU FUJITA
Department of Chemistry and
Materials Technology
Kyoto Institute of Technology
Matsugasaki, Sakyo-Ku,
Kyoto, 606-8585 Japan
fujitas@chem.kit.ac.jp

MICHEL GOOSSENS
CN Division
CERN
CH-1211 Geneva 23, Switzerland
m.goossens@cern.ch

ALAN HOENIG
17 Bay Avenue
Huntington, NY 11743, USA
+1 516 385-0736
ajhjj@cunyvm.cuny.edu
ahoenig@suffolk.lib.ny.us

STEPHANIE HOUGUE
AlphaSimplex Group
One Cambridge Center
9th Floor
Cambridge, MA 01242, USA
shogue@typewright.com

MIMI JETT
Institute for Advanced Learning
IBM Research
(use TUG Office address)
+1 503 578-2366
jett@us.ibm.com

JUDY JHONSON
jannejohnson@yahoo.com

DONALD E. KNUTH
Department of Computer Science
Stanford University
Stanford, CA 94305, USA

WENDY MCKAY
Control and Dynamical
Systems 107-81
California Institute of Technology
Pasadena, CA 91125, USA
wgm@cds.caltech.edu

PATRICIA MONOHON
University of California San Francisco
Dill Research Lab
3333 California Street, #415
San Francisco, CA 94118, USA
+1 415 502-2839
pmonohon@zimm.ucsf.edu

ROSS MOORE
Macquarie University
NSW 2109, Australia
ross@maths.mq.edu.au

STEPHEN MOYE
American Mathematical Society
201 Charles Street
Providence, RI 02904, USA
sgm@ams.org

ROLF NIEPRASCHK
Persiusstr. 12
10245 Berlin, Germany
niepraschk@ptb.de

CHRISTIAN OBRECHT
3, impasse Bellevue
89100 Paron, France
christian.obrecht@wanadoo.fr

ARTHUR OGAWA
40453 Cherokee Oaks Drive
Three Rivers, CA 93271, USA
+1 209 561-4585; Fax: +1 209 561-4584
ogawa@teleport.com

ALEXANDER PERLIS
Department of Mathematics
The University of Arizona
Tucson, AZ 85721, USA
aprl@math.arizona.edu

CHERYL PONCHIN
Center for Communications Research
Institute for Defense Analyses
29 Thanet Road
Princeton NJ 08540-3699, USA
cheryl1@ccr-p.ida.org

ROY PRESTON
4 Avon Wharf
Bridge Street
Christchurch
Dorset BH23 1DY, UK
preston@lds.co.uk
http://www_lds.co.uk/preston/

Denis Roegel
 LORIA
 Campus scientifique
 BP 239
 54506 Vandœuvre-lès-Nancy cedex
 France
 roegel@loria.fr
<http://www.loria.fr/~roegel/>

Kristoffer Høgsbro Rose
 IBM
 T. J. Watson Research Center
 30 Saw Mill River Road
 Hawthorne, NY 10532, USA
krisrose@us.ibm.com

Michael Sofka
 C&CT, VCC 309
 Rensselaer Polytechnic Institute
 110 8th Street
 Troy, NY 12180-3590, USA
sofkam@rpi.edu

D. P. Story
 Department of Theoretical and Applied Mathematics
 The University of Akron,
 Akron, OH 44278, USA
dpstory@uakron.edu

Nobuya Tanaka
 Department of Chemistry and Materials Technology,
 Kyoto Institute of Technology,
 Matsugasaki, Sakyo-Ku,
 Kyoto, 606-8585 Japan
nobuya@chem.kit.ac.jp

Philip Taylor
 The Computer Centre,
 Royal Holloway and Bedford New College,
 University of London,
 Egham Hill
 Egham, Surrey TW20 0EX, UK
P.Taylor@vax.rhbnc.ac.uk

Christina Thiele
 15 Wiltshire Circle
 Nepean K2J 4K9, Ontario Canada
cthiele@ccs.carleton.ca

David M. Tulett
 Faculty of Business Administration
 Memorial University of Newfoundland
 St. John's, NF, Canada, A1B 3X5
dtulett@mun.ca

Didier Verna
 EPITA, Research and Development Laboratory,
 14-16 rue Voltaire
 94276 Le Kremlin-Bicêtre,
 France
didier@lrde.epita.fr

Herbert Voß
 Wasgenstr. 21
 14129 Berlin, Germany
voss@perce.de

Jana Voß
 Wasgenstr. 21
 14129 Berlin, Germany
jana@perce.de

Peter Wilson
 18912 8th Ave. SW
 Normandy Park, WA 98166, USA
peter.r.wilson@boeing.com

Hermann Zapf
 Seitersweg 35
 D-64287 Darmstadt, Germany

Cartoon

by Roy Preston



General Delivery

From the Board of Directors

Arthur Ogawa
TUG Vice President

Editor's note: Although this issue of *TUGboat* has a cover date of December 2001, we're including this news as of the printing date, June 2003, in the interest of timely communication.

Greetings!

As of June 2003 the TeX Users Group has made significant progress working on problems of long standing, yet the changing times have brought us new challenges.

Here are some of the matters that have been effectively addressed.

- The TUG office is now competently run.

In July 1997, the TUG office was moved to its present location in Portland, Oregon, from San Francisco, California. Changes in location were stressful, but far more disruptive were four complete changes in office staff in the period March 1997–September 1998.

A further change was made at the beginning of 2000, when our managing consultant Richard Detwiler hired the current office manager, Robin Laakso. By dint of hard work, volunteer help, and aided by part-time employee Janice Carter, our accounting books are up to date, our member database is correct, and needed reports are being provided to the Board of Directors. In short, the office is well run.

It has long been recognized that the office manager, as the sole person paid to be concerned full time with TUG affairs, is at the heart of TUG. The Board is very pleased with Robin's performance and is committed to supporting her. It is a pleasure to work with her.

- The TeX Live and the Dante CTAN CD-ROMs now ship on time.

TUG's software offerings, updated yearly, used to be inserted into *TUGboat*, but are now shipped separately as they become available. The change reflects the fact that TUG members now perceive the CDs as a primary benefit of membership.

- TUG is a key player in TeX's Internet presence.

Thanks to a generous donation by Dante e.V. in 1996, TUG reinstated a North American node of the Comprehensive TeX Archive Network (CTAN), first

created by TUG member George Greenwade at Sam Houston State University, joining those in the UK (cam.ctan.org) and Germany (dante.ctan.org). TUG purchased new hardware for tug.ctan.org, since 2001 run by volunteer and TUG member Jim Hefferon (Saint Michael's College, Colchester, Vermont).

TUG also participates in the creation of TeX Live through its host tug.org (run by volunteer and TUG director Kaja Christiansen, University of Århus), as well as substantial efforts by many individual TUG members.

TUG hosts a number of TeX-related email lists by means of which volunteers coordinate their efforts in projects such as TeX Live, Web2C, LaTeX2HTML, texd, pdftex. In 2003, the original and still very much living TeX list texhax moved to tug.org. Archives and subscription information reside at <http://tug.org/mailman/listinfo/texhax>.

Finally, TUG supports TeX user groups worldwide by selling to them, at cost, the TeX Live CD-ROM. TUG provides this service to anyone wishing to purchase the CD in bulk.

In the future, your TUG membership will continue to support these activities with needed hardware purchases. And TUG is pleased to note that TUG members are among the many volunteers who work on CTAN, TeX Live, and the many other TeX-oriented projects and mainaining lists, all devoted to helping TeX users.

- TUG membership has recovered from its decline.

TUG members numbered about 1400 in 1997, the culmination of a gradually increasing slide starting about 1990. TUG membership has been stable for the last few years at about 2000.

The cause of the decline is not well understood, but my belief is that the cause of its reversal is the TeX Live CD-ROM. TUG shipped TeX Live 2 to all members in 1997 and has continued to do so each year since. The Dante CTAN CD-ROM was added as a benefit of membership in 1998.

- *TUGboat* now has a paid Managing Editor.

At its meeting in 2001, the TUG Board of Directors designated Robin Laakso as the *TUGboat* Managing Editor. As such, she assists Editor Barbara Beeton and the *TUGboat* Production Team in producing our journal. The hope is that *TUGboat* operations will benefit from the involvement of a designated managing editor, at the same time being more accountable and visible to the board.

This has proven to be the case, as three issues (counting this one) have shipped since February

2003. Also, there has proven to be a very useful synergy between Robin's work as TUG's office manager, and as *TUGboat*'s managing editor.

- TUG's IRS status has changed to 501(C)(3).

Up until recently, TUG's status vis à vis the US Internal Revenue Service had been 501(C)(6). As such TUG was a non-profit, but not considered a charitable organization: donations to TUG were not tax deductible.

Several years ago, TUG President Mimi Jett initiated an attempt to change our IRS status. Working with an attorney, we successfully applied to the IRS for the change; as of February 25, 2003, TUG was granted 501(C)(3) status, retroactive to September 10, 2002, to be reviewed in five years. (Please see <http://tug.org/tax-exempt>.)

The expected benefits are high: as a charitable organization, TUG is now much more likely to receive donations in money or in kind, since such donations would generally be tax-deductible (in the USA). The new status also makes it easier for corporate employees to attend our conferences.

Expect to see more about this in future *TUGboat* columns.

- TeX Users Group now supports technical development financially.

A recent initiative by TUG's Technical Council was to propose the formation of a TeX Development Fund. Now approved by the TUG Board, the Fund will award stipends to projects that will benefit TeX users; see <http://tug.org/tc/devfund/>. The Development Fund awarded its first round of grants in March 2003; a report will be printed in a future *TUGboat*.

New challenges face TUG

- *TUGboat* articles and conference papers are increasingly difficult to get.

Reports from both *TUGboat* editors and conference program committees agree that it is much more difficult to find the material to publish or present.

TUGboat had noted the developing problem as early as 1997. And as recently as 2000, conference papers seemed to be in abundance, but there appears to have been a shift here as well.

Notwithstanding, our efforts at soliciting material have had some success. About half the articles in this issue come from solicitations to authors making CTAN uploads. Volunteers to help with this (and other *TUGboat* work) would be most welcome, please email tugboat@tug.org if you'd like to help.

- *TUGboat* continues to publish, but about 12 months late.

Over the past five years, *TUGboat*'s production schedule has slipped to the point where it now ships about 12 months late. Some attribute the problem to the lack of papers submitted.

The TUG Business Committee recently approved a plan to combine *TUGboat* issues 1 and 2 into one printed issue; the first of these was Volume 21, Number 1/2, mailed out earlier this year. The expected benefit of this decision was to help bring *TUGboat* back on schedule. The TUG Board has recently passed on a motion to reduce the number of *TUGboat* issues from four to three, formalizing this change.

We expect the second and third *TUGboat* issues of 2002 to also be combined into a double issue, and we are now printing conference issues as they are produced (out of sequence). We hope that prompt publication of conference proceedings will be welcomed by our members.

- TUG membership has stagnated.

I noted above that TUG's membership has remained steady at about 2000 for a few years. This is better than a decline, but by comparison to, e.g., the German users group (at about 2200 members), it is certainly nothing to be complacent about.

In many respects, TUG is at an understandable disadvantage relative to some of the other user groups. But I think that TUG can improve its membership numbers. The Board has discussed forming a Membership Committee to do further work along these lines.

Earlier in 2003, we mailed a survey to 2001 members who did not renew in 2002. The results indicated that the biggest reason for non-renewal was lateness of *TUGboat*, which we are addressing as best we can, as indicated above.

- Relations with European TeX groups must be reviewed.

Volker Schaa, president of our sister organization in Germany, Dante e.V., wrote in May 2002 and again in October 2002 to the TUG Board of Directors with concerns shared by TeX user groups in Europe. He questioned the value of joint memberships and notified us of the European groups' intention to cooperate with each other on TeX conferences in future.

Also, Volker tells us that the European user groups have formulated a plan to hold joint conferences in Europe (EuroTeX) every year for the next five years. In the recent past, TUG has held conferences in Europe at Dubna (1996), Toruń (1998),

and Oxford (2000). The arrangement in future will probably have to change, given that the EU funding agency will require the conference to be that of a European group, not TUG.

Since fall 2002, we have been in communication with the European (and other) user groups, and have established grounds for cooperation on conference scheduling, technical development, and other matters.

TUG 2003 elections

Spring 2003 saw the election of a new president, longtime Director Karl Berry, and new members, Samuel Rhoads and Gerree Pecht, to the TUG Board of Directors. More information is at <http://tug.org/election/2003/results.html>.

The TUG Board is the steward of the organization, overseeing operations, addressing problems, implementing solutions, donating labor, expertise, enthusiasm, and support. At all times, we bear in mind the charter of the *T_EX* Users Group: to encourage the use of *T_EX* (and related programs) and to support *T_EX* users.

TUG 2003 in Hawai‘i

The 25th annual TUG Conference and Annual General Meeting will be held July 20–24, 2003, at the Outrigger Waikoloa Beach Resort, Hawai‘i. This event is already shaping up to be one of the most memorable conferences we have ever held, with special guest Duane Bibby, other eminent speakers, and exciting talks, all taking place at what must be the most lovely location around, the Kohala Coast of the Big Island of Hawai‘i.

Please refer to the conference’s web page at <http://tug.org/tug2003> for details and on-line registration form.

Are you interested in presenting your work in *T_EX* at one of our conferences? Please see <http://tug.org/tug2004>.

Your help is needed

Because TUG is an organization of volunteers, all of the benefits to our members come through the actions of people like you. Please consider how you can help the *T_EX* community!

- If you use *T_EX* in the workplace, perhaps your employer would like to become an institutional member or a sponsor.
- If you use *T_EX* professionally, or know others who do so, please remember that TUG provides key support for the creation and manufacture of the *T_EX* Live CD-ROM. Encourage people who use *T_EX* to look into the benefits of *T_EX* Live.

- If you know of or have helped develop an application of *T_EX* software, please consider writing a paper for *TUGboat* or presenting a paper at a *T_EX*-related conference.
- Consider donating some of your time to helping TUG put on conferences and training classes or publish *TUGboat*.
- If you are writing or producing a book using *T_EX*, please mention that fact in the colophon or acknowledgements, to help spread the word.
- For those with a technical bent, please help with the *T_EX* Live (texlive@tug.org) and CTAN projects. While not TUG-specific activities, both CTAN and *T_EX* Live are run entirely by volunteers, most of them members of *T_EX* user groups, and we support them wholeheartedly. *T_EX* Live in particular needs help with its installer and its documentation.
- If you have any ideas for encouraging the use of *T_EX* or developing TUG membership, please bring them forward.

The TUG board welcomes your input or questions at any time. Please contact us by email at board@tug.org.

◊ Arthur Ogawa
TUG Vice President
ogawa@teleport.com

Editorial Comments

Barbara Beeton

The status of *TUGboat*

This is the final issue of *TUGboat* for 2001. Thanks to everyone on the *TUGboat* production team for their hard work to make it happen.

Beginning with 2003, *TUGboat* will comprise three issues per year. This possibility was mentioned in my comments in the first 2001 issue (**22:1/2**). We hope that with fewer issues, we can attract articles of the high quality we strive for, and get the boat back on schedule.

The first 2002 issue has already appeared; it contains the proceedings of the 2002 TUG meeting. The next issue will be a combined issue (**23:2/3**); for this, we are planning on publishing two manuals: one about document creation, and the other on font installation. The last issue will be a “regular” issue (like this one), with articles on a variety of topics.

For 2003, the first issue will contain the proceedings of EuroTEX 2003. The second issue will contain the proceedings of the 2003 TUG meeting. And the third will again be a “regular” issue.

Glitches in *TUGboat* 22:1/2

Two articles in this year’s first issue had problems that escaped the notice of your editor.

In the first case, the glitch made it into print: a preliminary version of the translation of the “Laudatio for Professor Hermann Zapf” (pages 24–26) was published instead of the final version. Apologies to Frank Mittelbach. We also failed to identify the occasion for this presentation; it was delivered at the March 2000 meeting of Dante e.V., when Hermann Zapf was made an honorary member of the organization. The original, in German, appeared in *Die TeXnische Komödie* 1/2000, pages 31–36. The final version of the translation is posted on the *TUGboat* web site; look for the March/June 2001 issue linked from <http://www.tug.org/tugboat/contents.html>.

The second case is something that wasn’t printed: the table of contents entry for “Drawing message sequence charts with L^AT_EX”, by Sjouke Mauw and Victor Bos (pages 87–92) was omitted. Please write it in on your copy of the issue.

Zapfest exhibition, honoring Hermann Zapf

The opening of an exhibition in September 2001 at the San Francisco Public Library, “Calligraphic Type Design in the Digital Age: An Exhibition in Honor of the Contributions of Hermann and Gudrun Zapf”, was attended by both Hermann Zapf and his wife, Gudrun Zapf von Hesse, as well as by various TUG members in the area. The TeX Live CD and drawings for the Euler fonts were among the items exhibited.

Several photos of exhibited material, including the CD, can be seen at <http://www.cds.caltech.edu/~wgm/zapfest/>. Thanks to Cal Jackson (Caltech) for the photos.

Mordecai Richler font

The late Canadian author Mordecai Richler, whose best-known work is probably *The Apprenticeship of Duddy Kravitz*, has been honored by the creation of a typeface to be known by his name. The font was commissioned by The Giller Prize and Random House of Canada, in consultation with the Richler family. The font will become the official typeface of The Giller Prize and has been used to set Richler’s last book, *Dispatches from the Sporting Life*.

The font was created by Canadian type designer Nick Shinn. It is a classic book typeface, but with strong modern characteristics, including distinctive letters “M” and “R” that will serve to identify it. The typeface was announced both on CBC radio and in the Toronto *Globe & Mail*, an unusual recognition for a new font.

Details of the commission, Richler’s association with The Giller Prize, and the font can be found at <http://www.randomhouse.ca/richler>.

References for TeX and Friends

Peter Karp and Michael Wiedmann have announced the initial release of their ongoing documentation project, “References for TeX and Friends”. The purpose of this project is to provide help/reference files for L^AT_EX (and friends like ConTeXt, METAFONT, METAPOST, etc.) using the DocBook/XML format.

Various outputs can be generated from these source files, including formats for use with browsers and PDAs.

Contributions to this project in the form of either additional formats or documentation source are encouraged and welcome.

For information, see <http://www.miwie.org/tex-refs/>.

Institut d’Histoire du Livre

In September 2001, the Institute of the History of the Book held the inaugural session of their annual Book History Workshop. A lecture by the North American bookseller Bernard M. Rosenthal, entitled “The Gentle Invasion: Continental emigré booksellers of the thirties and forties and their impact on the antiquarian book trade in the United States” (“Quelques aspects du commerce du livre ancien en Europe et aux États-Unis aux 19e et 20e siècles”), is posted in both English and French on the IHL web site: <http://ihl.enssib.fr/>. Access it via the “archives” link.

More historic books online at the British Library

Joining the copies of the Gutenberg Bible mentioned previously in this column (22:1/2) are a number of other treasures, including the Lindisfarne Gospels (a manuscript from the early 8th century), Sultan Baybars’ Qur’ān (from the early 14th century), and the Tyndale New Testament (the first printed New Testament in English). Find them at <http://www.bl.uk/collections/treasures/digitisation.html>.

(An up-to-date installation of Shockwave is required in order to “turn the pages”.)

Web document analysis

The first International Workshop on Document Analysis was held in Seattle on 8 September 2001. The proceedings can be found online at <http://www.csc.liv.ac.uk/~wda2001/>.

The little tugboat that could

The photos at <http://koti.mbnet.fi/~soldier/towboat.htm> tell the truly amazing saga of a “real” tugboat that got itself into rough water. Don’t try this at home!

◊ Barbara Beeton
American Mathematical Society
P. O. Box 6248
Providence, RI 02940 USA
bnb@ams.org

ument is completed, it can be viewed by the Acrobat Reader 4.0 or higher.²

The technique is meant to be used in document development, preparation and assembly for authors who want to tap into the power methods of JavaScript.

2 Summary of the technique

The **execJS** technique consists to two parts:

1. A new L^AT_EX environment, the **execJS** environment
2. A few lines of folder-level JavaScript

The **execJS** environment is implemented as part of the **insdls Package**³, which was written in preparation for the T_EX Users Group 2001 Conference.

Within the **execJS** environment the “discardable” JavaScript is written to an auxiliary file with an extension of **.fdf**. When the newly created PDF document is opened for the first time in the viewer (either Acrobat or Approval) the **.fdf** file is imported into the document, and the JavaScript contained within the file is executed.

The second part of the technique, the folder-level JavaScript, gives the “discardable” JavaScript the right to execute security-restricted JavaScript methods.

3 Animated motivation

In the past few years, document authors who are producing interactive PDF documents from a L^AT_EX source have grown significantly in number.⁴ Many authors, myself included, crave to have the ability to use Acrobat’s powerful JavaScript interpreter to its fullest, all from a L^AT_EX source. The development of the **execJS** technique is a significant step in that direction. The **execJS** method came about by my pursuit of a holy grail of L^AT_EX/PDF production, *animation*. One of the examples that appears in this article is a PDF animation created entirely within a L^AT_EX source file!

4 The execJS technique

Acrobat Version 5.0 comes with an extended FDF (Forms Data Format) specification. This specification creates a new **Doc** key, the value of which refers to JavaScript contained within the FDF file that is to

² Version 5.0 or later of Reader is required if JavaScript objects, properties or methods are used not available in version 4.0.

³ The **insdls Package**, a standalone package, is distributed as a component package of the AcroTeX eDucation Bundle: <http://www.math.uakron.edu/~dpstory/webeq.html>

⁴ For links to and descriptions of some of the many authors doing quality work on the Web, see the AcroTeX web site: <http://www.math.uakron.edu/~dpstory/acrotex.html>

Electronic Documents

execJS: A new technique for introducing discardable JavaScript into a PDF file from a L^AT_EX source

D. P. Story

1 Introduction

This article describes a technique, referred to as **execJS**¹, for writing JavaScript from within a L^AT_EX source file that will be executed once when the document is first opened, then *discarded*. The **execJS** technique allows you to execute JavaScript methods, *even ones that have their use restricted for security reasons*. The discardable code and the ability to execute even security-restricted methods are the distinguishing features of **execJS**.

The method requires Acrobat 5.0 or later (the full application, not the Acrobat Reader), or Acrobat Approval 5.0 or later. You can, though not required, use the Acrobat Distiller to create the PDF document; or, as so many do, you can use either the **pdftex** or **dvipdfm** applications. Once the PDF doc-

¹ execute JavaScript

be imported into the document as Document Level JavaScript (DLJS). See the paper (Story, 2001) for details of how to use this specification to insert DLJS from a L^AT_EX source. DLJS can be inserted from your L^AT_EX source file using the `insdljs` Package. The specification also defines an `After` key, the value of which is an indirect reference to JavaScript code also contained in the FDF file.

The `After` key is the one of interest in this paper. The JavaScript referenced by the `After` key is executed after the FDF is imported into the document. The JavaScript is executed *but not saved*, it is “discardable”.

Consider the following FDF file containing both the `Doc` and `After` keys.

```
%FDF-1.2
1 0 obj
<< /FDF
  << /JavaScript
    << /Doc 2 0 R /After 3 0 R
    >>
  >>
endobj
2 0 obj
[ (ExecJS execjs) (var _execjs = true;) ]
endobj
3 0 obj
<<>>
stream
app.alert("Discardable\ code executed!");
endstream
endobj
trailer
<< /Root 1 0 R >>
%%EOF
```

When this file is imported into Acrobat (or Approval), one line of code will be placed at the document level, `var _execjs = true`, and one line of code

```
app.alert("Discardable\ code executed!");
will be executed, but not saved.
```

4.1 The execJS environment

It was a simple modification of the `insdljs` Package to implement an `execJS` environment that writes a FDF file containing the `After` key. Thus,

```
\usepackage[execJS]{insdljs}
...
\begin{execJS}{execjs}
app.alert("Discardable\ code executed!");
\end{execJS}
```

writes the file seen in the FDF above. The required argument for this environment is the base file name of the FDF to be written.

The environment not only writes the FDF file, if the `execJS` option of the `insdljs` Package is spec-

ified, it also adds an open page action to the first page of the document:

```
if (typeof _execjs == "undefined")
  this.importAnFDF("execjs.fdf");
```

This is a point in the whole technique where Acrobat or Approval is required. The JavaScript method `this.importAnFDF()` is not available for the Acrobat Reader.

When the document is opened for the first time, the file `execjs.fdf` gets imported into the PDF document. The FDF also inserts the DLJS `var _execjs = true` (which will be saved with the document). This variable declaration prevents the FDF from being repeatedly imported each time the first page is viewed.

4.2 Folder JavaScript

The `execJS` environment writes the FDF file which, in turn, is imported into the newly created PDF, the JavaScript is executed, and is “discarded” (not saved); however, security-restricted JavaScript still cannot be executed. An additional trick is needed.

Many of the restricted methods can be used only during, what Acrobat calls, menu, console or batch events. The approach taken here is to execute JavaScript through a menu event. To do this, create a file, called `myJS.js` (or any name with an extension of `.js`). The contents of this file are the lines:

```
_MenuProc = function() {};
app.addMenuItem({
  cName: "MenuProc",
  cUser: "Menu Procedure",
  cParent: "Tools",
  cExec: "_MenuProc()", 
  nPos: 0
});
```

The first line defines `_MenuProc`, a JavaScript function which does nothing. The rest of the lines define a new menu item under the “Tools” menu. Additional menu items can only be created through a folder level JavaScript file.

Place `myJS.js` in the `JavaScripts` folder, follow the path `Acrobat 5.0/Acrobat/JavaScripts`. This is the folder in which `aform.js`, the folder level JavaScript for the forms plugin, is kept.

The trick for executing JavaScript methods that are restricted to a menu event is to redefine the Folder level function `_MenuProc`. Consider the following code within the `execJS` environment:

```
\begin{execJS}{execjs}
function importMyIcons ()
{
  for ( var i=0; i < 36; i++)
    this.importIcon("rotate"+i,"animation.pdf",i);
```

```

}
_MenuProc = importMyIcons;
app.execMenuItem("MenuProc");
_MenuProc = function() {}
\end{execJS}

```

A function `importMyIcons` is defined that imports a series of named icons using the `this.importIcon()`. The use of this method is restricted to menu, console or batch events if a file name is given as the second argument. The first argument is the name of the icon newly embedded in the document.

Following the function definition, we make the assignment

```
_MenuProc = importMyIcons;
```

then execute the menu item named "MenuProc", which, in turn, causes `_MenuProc` (now assigned as `importMyIcons`) to be executed. (Multiple function definitions and assignments can be made in this way.) Finally, the `_MenuProc` function is reassigned its default definition (optional).

Important: After the document is assembled, save the document (as you can with Acrobat and Approval), this will save the DLJS, but not the "discardable" code.

See the Acrobat JavaScript Object Specification, Version 5.0 or later,⁵ for details of the JavaScript methods just illustrated.

5 An animation example

In this section, an animation example is presented. The animation is done completely within the L^AT_EX source file. After the PDF document is finally assembled, the animation is ready to run.

The animation was done using two L^AT_EX files, `execjstst.tex` and `animation.tex`. The AcroTeX eEducation Bundle⁶ was used. The AcroTeX Bundle includes the following packages:

- The Web Package: For creating good looking PDF documents for the Web.
- The Exerquiz Package: For creating online exercises and quizzes. (In the animation, the form field macros only were used.) Exerquiz loads the `insdljs` Package and passes the options of `insdljs` to it.
- The `insdljs` Package: A package used to insert Document Level JavaScripts, to create open actions, and to create immediately executable and "discardable" JavaScript.

⁵ Available under the Help menu of Acrobat, or at <http://partners.adobe.com/asn/developer/technotes/acrobatpdf.html>

⁶ Available at CTAN:/tex-archive/macros/latex/contrib/supported/webeq or at the Bundle's homepage <http://www.math.uakron.edu/~dpstory/webeq.html>

- The `dljslib` Package: A library of useful JavaScript functions that can be "checked out" for use.

5.1 animation.tex

The `animation.tex` file is the L^AT_EX source used to create the series of images that will be displayed in the animation. This animation is rather simple. PSTricks was used to create a series of 36 graphics:

```

\documentclass{article}
\usepackage{pstricks,pst-plot}
\usepackage[dvipsone]{web} %<- or dvips

\margins{0pt}{0pt}{0pt}{0pt} % no margins
\screensize{1in}{1in} % 1in by 1in

\pagestyle{empty}\parindent=0pt
\SpecialCoor

\begin{document}
\psset{unit=1in,origin={-.5,-.5}}

\multido{\i=90+-10}{36}{%
\begin{pspicture}(1in, 1in)
\psframe[fillstyle=solid,
          fillcolor=lightgray,
          linecolor=lightgray](-.5,-.5)(.5,.5)
\psline[linecolor=blue](0,0)(.5; \i)
\pscircle[linecolor=red](0,0){.5}
\end{pspicture}
\newpage
}
\end{document}

```

This particular file was distilled to create the PDF document, `animation.pdf`. For more sophisticated animations, the graphic images need to be created using some high-end application.

5.2 ExecJStst.tex

The file `execjstst.tex` is actually the animation demo. Here is the verbatim listing of this file.

```

\documentclass{article}
\usepackage
  [dvipsone, % <- dvips, pdftex, dvipdfm
   designi,
   ]{web}
% exerquiz loads insdljs, and passes execJS to it
\usepackage[execJS]{exerquiz}

% Embed the animation images as named icons, the ith
% icon is named rotatei, i=0..35
\begin{execJS}{execjs}
function importMyIcons ()
{
  for ( var i=0; i < 36; i++)
    this.importIcon("rotate"+i,"animation.pdf",i);
}
_MenuProc = importMyIcons;
app.execMenuItem("MenuProc");
_MenuProc = function() {}
\end{execJS}

% Define a JavaScript action that will be attached

```

```
% to the button "aniCtrl" that starts the
% animation.
\newcommand{\aniCtrlAction}
{%
/A << /S /JavaScript /JS
%
  function ShowIt()\jsR
{\jsR\jsT
  var oIcon;\jsR\jsT
  oIcon=this.getIcon
  ("rotate"+run.count);\jsR\jsT
  f.buttonSetIcon( oIcon, 0);\jsR\jsT
  run.count++; \jsR\jsT
  run.count \%/= 36;\jsR
}\jsR
var f = this.getField("myAnimation");\jsR
var run = app.setInterval("ShowIt()",100);\jsR
run.count = 0;\jsR
var timeout = app.setTimeout
  ("app.clearInterval(run);", 3*3600+200);
) >
}

\begin{document}
\section*{An Animation}

% Center the animation "window" and the start button
\begin{center}
\eqIcon{myAnimation}{72bp}{72bp} \\
\eqGenButton[\CA{Push}]{\rawPDF{\aniCtrlAction}}
{aniCtrl}{36bp}{16bp}
\end{center}
\end{document}
```

Many of the above commands come from the *exerquiz* Package. The macros `\jsR` and `\jsT` expand to `\r` and `\t`, which are escape sequences in JavaScript for newline and tab; `\eqIcon` makes it easy to create a button field set up for displaying a (PDF) icon as its face appearance; `\eqGenButton` is a general command for creating push buttons.

The JavaScript action defined by L^AT_EX command, `\aniCtrlAction`, first defines a JavaScript function called `ShowIt`, which gets the *icon object* using the `getIcon` method, sets the button appearance of the `myAnimation` field using the field method `buttonSetIcon` method, and increments a counter; at the top-level, the script gets the *field object* for the `myAnimation` field, starts a timing event which calls the function `ShowIt` every 100 milliseconds, initializes the counter, and sets the timeout interval.

6 Another example

Recently, I produced a PDF version of an `html` online survey for the **Seybold PDF Conference 2002**. The online version was an “intelligent” survey: the answer to one question determined what question would next be posed to the respondent.

L^AT_EX and the AcroTeX Bundle were used to duplicate the questions and the functionality of the `html` version in PDF. A short macro package was

written, DLJS was automatically introduced into the document using the `insdljs` package which supported the “logic” of the survey.

The PDF version uses form templates. (A form template is a special type of page that can be hidden or made visible, a copy of a template can be spawned, Acrobat viewer or Acrobat Approval required.) Templates were used to reveal the next page, based on the answers given on the current page by the respondent.

Templates cannot be created using the `pdfmark` operator (`pdftex` and `dvipdfm` do not support their creation), but can be created using certain security-restricted JavaScript methods. The `execJS` technique was used to create the templates. As a result, a fairly complex PDF document was assembled entirely from the content and commands in the L^AT_EX source file.

Read the article “Seybold PDF survey in PDF also worthy of study”⁷ by Kurt Foss, Planet PDF Editor⁸, for more details of the operational capability of the PDF version of the survey.

7 In conclusion

L^AT_EX has become a powerful markup language for creating interactive PDF documents; in my opinion, the L^AT_EX system is *the premier* system for building, debugging, modifying and assembling complex PDF documents. The `hyperref` and `exerquiz` packages can be used to create links and form fields with JavaScript actions attached, the `insdljs` package allows for the insertion of document level JavaScript into the PDF document, open actions, and now executable, “discardable” JavaScript that can be used in a post-creation setting.

References

- Story, D. P. “Techniques of introducing document-level JavaScript into a PDF file from a L^AT_EX source”. *TUGboat* **22**(3), 161–167, 2001.

◇ D. P. Story
 Department of Theoretical and
 Applied Mathematics
 The University of Akron
 Akron, OH 44278
 dpstory@uakron.edu
 http://www.math.uakron.edu/~dpstory/

⁷ www.planetpdf.com/mainpage.asp?webpageid=2130

⁸ Planet PDF: www.planetpdf.com

L^AT_EX, SVG, Fonts

Michel Goossens and Vesa Sivunen

Abstract

After giving a short overview of SVG, pointing out its advantages for describing in a portable way the graphics content of electronic documents, we show how we converted T_EX font outlines (the Type 1 variant) into SVG outlines and explain how these SVG font glyphs can be used in SVG instances of documents typeset with T_EX.

1 Introduction

The increasing affordability of the personal computer drastically reduces the production cost of electronic documents. The World Wide Web makes distributing these documents worldwide cheap, easy, and fast. Taken together, these two developments have considerably changed the economic factors controlling the generation, maintenance, and dissemination of electronic documents. More recently, the development of the XML family of standards and the ubiquity of the platform-independent Java language make it possible to have a unified approach to handle the huge amount of information stored electronically and to transform it into various customizable presentation forms.

Given the severe financial constraints in many parts of the world, where it is often out of the question to even consider printing multiple copies of a (highly technical) document, electronic dissemination via the Web is the only way to publish. Thus, the Web is not only an additional medium for the traditional publishing industry, but a necessary complement in large parts of the world to participate in sharing scientific and technical information and benefit from the wealth and progress it creates.

Various techniques are now available to transform L^AT_EX documents into PDF, HTML (XHTML), or XML so that the information can be made available on the Web. Thus, L^AT_EX will continue to play a major role in the integrated worldwide cyberspace, especially in the area of scientific documents. However, it is clear that L^AT_EX's greatest impact will remain in the area of typesetting, with T_EX remaining an important intermediate format for generating high-quality printable PDF output.

The present article explores ways of transforming L^AT_EX-encoded information globally into a Scalable Vector Graphics (SVG) format, in particular by exploiting the use of the SVG font machinery. A further step, to be described in a forthcoming article, will be to transform the L^AT_EX document into vari-

ous XML vocabularies, thus saving as much semantic information as possible.¹ Such a modular approach makes optimal document reuse possible.

2 SVG for portable graphics on the Web

As the Web has grown in popularity and complexity, users and content providers wanted ever better and more precise graphical rendering, as well as dynamic Web sites. Today, only drop shadows, rudimentary animations, and low-resolution GIF or PNG images are commonly used in Web pages. Moreover, that technology is not really scalable.

The publication of the SVG Recommendation was the result of more than two years of collaborative effort by major players in the computer industry² to find a workable cross-platform solution to Web imaging. Version 1.0 of the SVG specification was published as a W3C Recommendation on 4 September 2001 and it represents a genuine advance for portable graphics on the Web. The current version of SVG is 1.1, and it became a W3C Recommendation on 14 January 2003.³

Nowadays many software vendors support SVG in their products, while more and more free viewing and editing tools capable of handling SVG become available.

SVG is an open-standard vector graphics language for describing two-dimensional graphics using XML syntax. It lets you produce Web pages containing high-resolution computer graphics.

SVG has the usual vector graphics functions. Its fundamental primitive is the *graphics object*, whose model contains the following:

- graphics paths consisting of polylines, Bézier curves, etc.:
 - simple or compound, closed or open;
 - (gradient) filled, (gradient) stroked;
 - can be used for clipping;
 - can be used for building common geometric shapes;
- patterns and markers;
- templates and symbol libraries;

¹ For instance, the hierarchical structure of the document is encoded in XML by using one of DocBook, TEI or, to a certain extent, XHTML, while other specific XML vocabularies are used for their given application domain, such as MathML for mathematics, SVG for two-dimensional graphics, CML for chemistry, BSML for bioinformatics, GeneXML or GEML for gene expression, and many others.

² Among the companies represented on W3C's SVG committee were IBM, Microsoft, Apple, Xerox, Sun Microsystems, Hewlett-Packard, Netscape, Corel, Adobe, Quark, and Macromedia.

³ *Scalable Vector Graphics (SVG) 1.1 Specification*, available at <http://www.w3.org/TR/SVG11/>.

- transformations:
 - default coordinate system: x is right, y is down, one unit is one pixel;
 - viewport maps an area in world coordinates to an area on screen;
 - transformations alter the coordinate system (2×3 transformation matrix for computers; translate, rotate, scale, skew for humans);
 - can be nested;
- inclusion of bitmap or raster images;
- clipping, filter and raster effects, alpha masks;
- animations, scripts, and extensions;
- groupings and styles;
- SVG fonts (independent from fonts installed on the system).

SVG consists of Unicode text in any XML namespace.⁴ The use of Unicode throughout enhances searchability and accessibility of the SVG graphics.

SVG drawings can be dynamic and interactive. The Document Object Model (DOM) for SVG allows for efficient vector graphics animation via scripting, which can be performed on SVG elements and other XML elements from different namespaces simultaneously within the same Web page. Event handlers can be assigned to any SVG graphical object.

A major source of information on SVG is the W3C SVG site, which describes the latest developments in the area of SVG, the status of current implementations. It also has a reference list of articles, books, software announcements, and pointers to other interesting SVG sites.⁵

2.1 Inside an SVG document

As described earlier, the basis of SVG is a Unicode text *document*, usually identified with a file extension `.svg` and a mime-type (for the server) `image/svg+xml`. Thus it is rather straightforward to create and edit SVG documents with your favorite text editor.

The top part of Figure 1 shows a small SVG file `svgexa.svg`, which is an example of the static

⁴ The target application must be able to interpret the specific XML vocabulary to make this useful. A forthcoming article will show how one can use XHTML, MathML and SVG together.

⁵ The W3C SVG site is at <http://www.w3.org/Graphics/SVG/>. An SVG Tutorial site is at <http://www.svgtutorial.com/>. The Batik distribution (<http://xml.apache.org/batik>) comes with many SVG examples, including quasi 3D scenes, animations, complex languages, such as Arabic, etc. The Adobe SVG site (<http://www.adobe.com/svg>) features some interesting SVG files. Interactive geometry, statistical charts, cartographic material and much more is at Michel Pilat's site (<http://pilat.free.fr/english>).

graphics possibilities of SVG. After the comment on line 1 we declare that we work in the SVG namespace (line 2) and define the size of the display area (line 3). We write a title (lines 5 and 6), draw a row of four rectangles (lines 7–14), followed by a row of four rectangles with rounded corners (lines 17–23), and finally a row of four ellipses (lines 25–29). The origin of SVG's x - y coordinate system is the upper left hand side of the display area. The semantics of the various arguments of the SVG elements should be rather easy to guess (the SVG Specification contains detailed definitions). Notice the similarity between the PostScript and SVG languages.

If we want to view our file `svgexa.svg` we must use an application that understands the SVG language, such as Apache's Batik⁶ or Adobe's Browser plugin `svgview`.⁷ W3C's browser Amaya,⁸ which provides an interesting development environment for viewing and editing MathML and HTML code, also supports part of SVG. The recent default distributions (1.3 or later) of the Mozilla browser⁹ have built-in support for presentation MathML but to be able to interpret SVG natively you need to download a special SVG-enabled executable.¹⁰ The middle row of Figure 1 represents at the left our example file as displayed by Batik, with a zoom on part of the graphics at its right. This shows that regions of an image can be magnified without loss of quality (this is because of the vector nature of SVG's graphics model).¹¹ The bottom row of Figure 1 shows at the left the same file as displayed by Microsoft's Internet Explorer using Adobe's `svgview` plugin installed (left) and at the right as shown by the Amaya browser.

3 Generating SVG instances from **TEX** fonts

Many scientific documents, especially in physics and mathematics, are marked up with **LATEX**. On the other hand XML has become a *lingua franca* of the Internet and XML-aware tools are becoming ubiquitous. Therefore, it becomes important to integrate **LATEX** and XML in an optimal way.

⁶ See <http://xml.apache.org/batik>.

⁷ Available from <http://www.adobe.com/svg>.

⁸ See <http://www.w3.org/Amaya/>.

⁹ See <http://www.mozilla.org>.

¹⁰ Details at <http://www.mozilla.org/projects/svg/>.

¹¹ The Batik Squiggle SVG viewer and Adobe's Browser plugin `svgview` offer convenient ways to navigate an image. You can zoom in and out, move in the two-dimensional plane (in `svgview` hold down the Alt or in Squiggle the Shift key and move the mouse with the left button pressed) or rotate over a given angle.

```

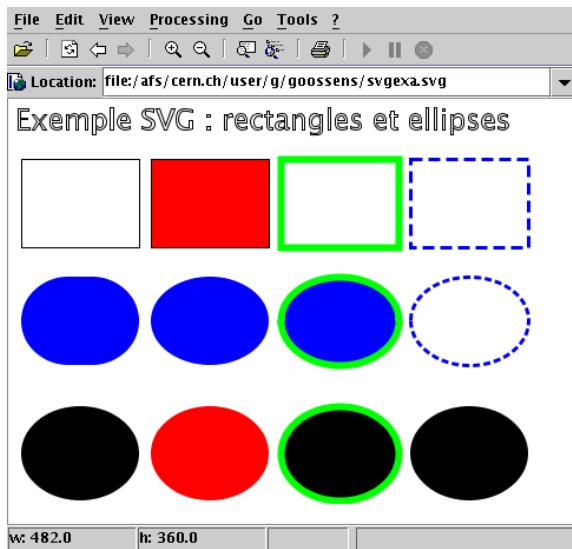
1 <!-- svgexa.svg --> A small SVG Example -->
2 <svg xmlns="http://www.w3.org/2000/svg"
3   width="450pt" height="350pt">
4   <g style="stroke:black; fill:none">
5     <text x="5" y="25" style="font-size:24">
6       Exemple SVG : rectangles et ellipses</text>
7     <rect x="10" y="50" width="100" height="75"/>
8     <rect x="120" y="50" width="100" height="75"
9       style="fill:red"/>
10    <rect x="230" y="50" width="100" height="75"
11      style="stroke:lime; stroke-width:6"/>
12    <rect x="340" y="50" width="100" height="75"
13      style="stroke:blue; fill:none; stroke-width:3;
14        stroke-dasharray:10 5;stroke-linejoin:miter"/>
15  </g>

```

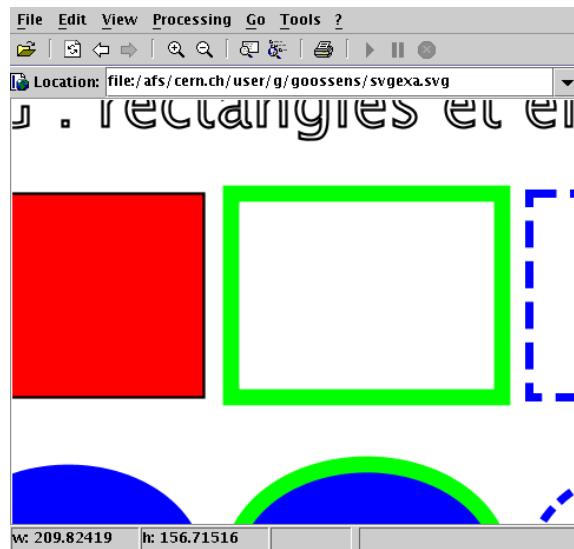
```

16   <g style="stroke:none; fill:blue">
17     <rect x="10" y="150" width="100" height="75" rx="40" ry="40"/>
18     <rect x="120" y="150" width="100" height="75" rx="50" ry="50"/>
19     <rect x="230" y="150" width="100" height="75" rx="60" ry="60"
20       style="stroke:lime; stroke-width:6"/>
21     <rect x="340" y="150" width="100" height="75" rx="70" ry="70"
22       style="stroke:blue; fill:none; stroke-width:3;
23         stroke-dasharray:6 3;stroke-linejoin:miter"/>
24   </g>
25   <ellipse cx="60" cy="300" rx="50" ry="40"/>
26   <ellipse cx="170" cy="300" rx="50" ry="40" style="fill:red"/>
27   <ellipse cx="280" cy="300" rx="50" ry="40"
28     style="stroke:lime; stroke-width:6"/>
29   <ellipse cx="390" cy="300" rx="50" ry="40" angle="45"/>
30 </svg>

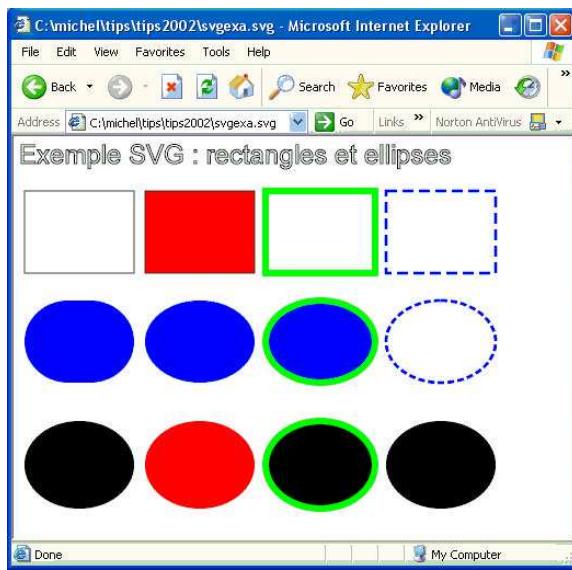
```



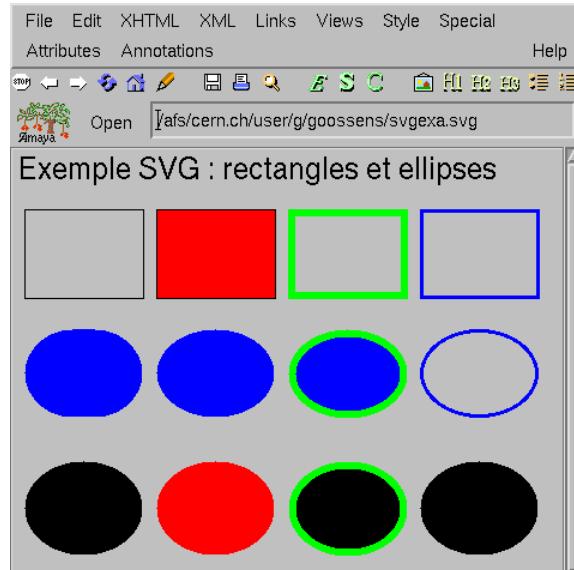
(a) Batik Squiggle SVG viewer



(b) Zooming with Batik Squiggle



(c) Microsoft Explorer



(d) W3C's Amaya

Figure 1: Browsing an SVG file

SVG is a static declarative XML vocabulary; it provides only a final-form two-dimensional representation of a graphics image.¹² Thus paragraphs or pages must be formatted by upstream application programs (e.g., TeX, drawing tools, Java) or by inline escaping to scripting (in Perl, Python, Ruby, JavaScript, etc.)

Various tools exist to translate EPS files to SVG, e.g., Adobe's **Illustrator** (commercial) or Wolfgang Glunz' **pstoedit**¹³ (the SVG translator option is shareware). For direct translation from DVI there is Adrian Frischaufer's **dvi2svg**.¹⁴

The translations work quite well as long as one uses standard fonts, such as Times, Helvetica and Courier. However, these applications have problems with TeX's non-standard character font encodings.

This problem of font encoding is closely related to the fact that SVG, being an XML language, uses Unicode as basic character encoding. So if one wants to go from TeX (DVI) to SVG the driver has to map each TeX character to its Unicode code-point and use a *large* corresponding SVG font that encodes all the needed characters.¹⁵ Such a full mapping, although not impossible, is far from trivial. Hence, in coordination with Glunz, we have opted for a temporary hack, where we use a special *ad-hoc* option for **pstoedit**, where we map for each font instance the 256 hexadecimal codes 00 to FF in the TeX font encoding into Unicode's "Private Use Area" (PUA) using (hexadecimal) code positions E000 to E0FF.

3.1 Producing SVG font instances

If we want to work with Computer Modern METAFONT sources we can use Szabó Péter's **TeXtrace** program.¹⁶ It is a collection of Unix scripts that convert any TeX font into a Type 1 .pfb outline font¹⁷ that is immediately suitable for use with **dvips**, **pdftex**, Adobe's **acroread**, etc. It now also has an option to generate SVG, but this did not directly do what we needed. In fact, we preferred to use **TeXtrace** only to generate Type 1 pfb files, and fall back on the second approach, that we describe next.

Working from Type 1 pfa/pfb font sources it is rather straightforward to generate the corresponding SVG font. A Perl script, **t1svg.pl**, was developed to achieve this translation, where the only (mi-

¹² This is unlike the PostScript language, which allows for inline computations.

¹³ See <http://www.pstoedit.net/pstoedit>.

¹⁴ See <http://www.activemath.org/~adrianf/dvi2svg/>.

¹⁵ In fact, some mathematical TeX characters are still absent from and thus not yet encodable in Unicode 3.2.

¹⁶ See <http://www.inf.bme.hu/~pts/textrace/>.

¹⁷ See http://partners.adobe.com/asn/developer/pdfs/TN/T1_SPEC.PDF for a description of the Type 1 font format.

nor) difficulty is the correct handling of the position of the current point.¹⁸

Table 1 shows the correspondence between the PostScript Type 1 operators (left column) and the SVG equivalent (middle column), with the arguments for each command expressed in function of those of the corresponding Type 1 ones (a1, a2, etc.). The right column shows the *x* (cx) and *y* (cy) coordinates of the current point expressed in function of the Type 1 arguments.

3.2 Disassembling Type 1 font sources

Type 1 fonts are often commercial and should not be converted into SVG without the permission of the rights holder. TeX fonts, however, are publicly available. For TeX fonts that are not yet available in Type 1 format one can use Szabó Péter's **TeXtrace** program mentioned previously. Our Perl script needs the binary compressed pfb outline font to be disassembled with Lee Hetherington's and Eddie Kohler's **t1disasm** program into human-readable form.¹⁹

3.3 The conversion script

The Perl script **t1svg.pl**, which converts Type 1 fonts to SVG fonts, is relatively short and simple. It does not require any special modules and should run with any Perl version.

t1svg.pl reads the disassembled pfb file as input and translates the Type 1 operators into their SVG equivalents according to the correspondences of Table 1. Section A provides a detailed description of the SVG commands used in that Table.

The example of Table 2 shows the Type 1 source (left) and the SVG source (right) of the glyph of a contour integral. Line 1 of the SVG instance specifies the Unicode code position of the glyph (hexadecimal E049, i.e. in the PUA) and its name, which we copied from line 1 of the Type 1 source at the left (**contintegraldisplay**). Line 2 of the SVG specifies the horizontal width of the glyph (555.6). It was calculated by taking the second argument of the **hsbw** operator on line 2 of the Type 1 source, i.e., 5000 9 **div**, which divides 5000 by 9.²⁰ Then, on

¹⁸ This is needed since the SVG **z** operator sets the current point to the point that closes the path (after the operation), while the Type 1 **closepath** operator leaves it untouched, i.e., the current point remains at the value it had before the call. Therefore our Perl script has to keep track of the current point to restore it to the correct value with respect to the Type 1 coordinates after each **z** operator, at which point we insert an absolute **Moveto** (M) to the required (saved) current point coordinates before continuing.

¹⁹ See <http://www.lcdf.org/eddieltwo/type/#t1utils>.

²⁰ Remember that PostScript uses reverse Polish notation, with arguments preceding the operator to which they belong.

Table 1: Correspondence between Type 1 and SVG commands

Type 1	SVG	Current point
a1 a2 hsbw	horiz-adv-x = a2 (argument of glyph), M a1 0	cx=a1
a1 hlineto	h a1	cx=cx+a1
a1 hmoveto	m a1 0	cx=cx+a1
a1 a2 a3 a4 hvcurveto	c a1 0 (a1+a2) a3 (a1+a2) (a3+a4)	cx=cx+a1+a2, cy=cy+a3+a4
a1 a2 rlineto	l a1 a2	cx=cx+a1, cy=cy+a2
a1 a2 rmoveto	m a1 a2	cx=cx+a1, cy=cy+a2
a1 a2 a3 a4 a5 a6 rrcurveto	c a1 a2 (a1+a3) (a2+a4) (a1+a3+a5) (a2+a4+a6)	cx=cx+a1+a3+a5, cy=cy+a2+a4+a6
a1 vlineto	v a1	cy=cy+a1
a2 vmoveto	m 0 a1	cy=cy+a1
a1 a2 a3 a4 vhcurveto	c 0 a1 a2 (a1+a3) (a2+a4) (a1+a3)	cx=cx+a2+a4, cy=cy+a1+a3
closepath	z	

Table 2: Type 1 and SVG sources compared

```

1 /contintegraldisplay {
2 56 5000 9 div hsbw
3 -2222 22 hstem -2173 94 hstem -1381 40 hstem
4 -881 40 hstem -143 94 hstem -22 22 hstem
5 0 97 vstem 195 40 vstem 694 40 vstem 790 97 vstem
6 48 -2177 rmoveto 32 2 17 22 0 25 rrcurveto
7 33 -25 16 -23 vhcurveto -24 -25 -15 -35 hvcurveto
8 -51 50 -42 61 vhcurveto 155 0 58 242 76 322 rrcurveto
9 15 61 35 154 14 62 rrcurveto 151 119 122 148 hvcurveto
10 0 76 -35 117 -132 57 rrcurveto 30 179 31 184 37 173 rrcurveto
11 22 100 44 203 49 0 rrcurveto 31 0 25 -19 4 -4 rrcurveto
12 -33 -2 -17 -22 0 -25 rrcurveto -33 25 -16 23 vhcurveto
13 24 25 15 35 vhcurveto 54 -54 39 -55 vhcurveto
14 -79 0 -55 -117 -53 -197 rrcurveto
15 -22 -81 -34 -144 -6 -25 rrcurveto
16 -15 -61 -35 -154 -14 -62 rrcurveto
17 -151 -119 -122 -148 hvcurveto
18 0 -76 35 -117 132 -57 rrcurveto
19 -42 -253 -35 -196 -30 -123 rrcurveto
20 -18 -74 -46 -193 -82 0 rrcurveto -36 -25 23 0 hvcurveto
21 closepath
22 320 857 rmoveto -79 37 -54 80 0 92 rrcurveto
23 0 113 84 109 137 8 rrcurveto
24 closepath
25 104 -21 rmoveto 78 -35 56 -80 0 -94 rrcurveto
26 0 -113 -84 -109 -137 -8 rrcurveto
27 closepath
28 endchar
29 }
1 <glyph unicode="&#xE049;" glyph-name="contintegraldisplay"
2 horiz-adv-x="555.6">
3 <path style="fill:#000000; fill-rule=evenodd; stroke:none"
4 d="M 56 0
5 m 48 -2177
6 c 32 2 49 24 49 49
7 c 0 33 -25 49 -48 49 c -24 0 -49 -15 -49 -50
8 c 0 -51 50 -93 111 -93 c 155 0 213 242 289 564
9 c 15 61 50 215 64 277 c 151 0 270 122 270 270
10 c 0 76 -35 193 -167 250 c 30 179 61 363 98 536
11 c 22 100 66 303 115 303 c 31 0 56 -19 60 -23
12 c -33 -2 -50 -24 -50 -49 c 0 -33 25 -49 48 -49
13 c 24 0 49 15 49 50 c 0 54 -54 93 -109 93
14 c -79 0 -134 -117 -187 -314
15 c -22 -81 -56 -225 -62 -250
16 c -15 -61 -50 -215 -64 -277
17 c -151 0 -270 -122 -270 -270
18 c 0 -76 35 -193 167 -250
19 c -42 -253 -77 -449 -107 -572
20 c -18 -74 -64 -267 -146 -267 c -36 0 -61 23 -61 23
21 z
22 M 104 -2177 m 320 857 c -79 37 -133 117 -133 209
23 c 0 113 84 222 221 230
24 z
25 M 512 -881 m 104 -21 c 78 -35 134 -115 134 -209
26 c 0 -113 -84 -222 -221 -230
27 z"
28 />
29 </glyph>
```

line 4 of the SVG we have an absolute `move` (`M 56 0`), which corresponds to the first argument of the `hsbw` operator on line 2 of the Type 1 source. We ignore all of hints in the Type 1 source (lines 3 to 5) and continue with the `rmoveto` operator and its two arguments (line 6) which we find back in the SVG instance on line 5. We leave it as an exercise to the reader to walk through the remaining lines of the Type 1 source and verify, referring to Table 1 as necessary, that you indeed obtain the result shown

at the right. Note in particular how we have to reset the current point after each SVG `z` operator (i.e., the absolute move `M` on lines 22 and 25) to where it was before the `z` operator was executed.

Figure 2 shows the contour integral as viewed with `ghostview` (the PostScript Type 1 image) and with `Batik` (rendering the calculated SVG instance).

Rather than use the temporary hack to map the `TEX` code positions into the Unicode PUA it might

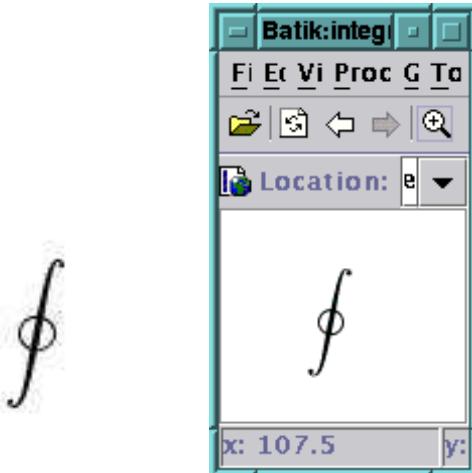


Figure 2: PostScript Type 1 and SVG rendering

be useful to map the \TeX characters to their correct position in the Unicode character space to obtain a large Unicode-encoded font. This would allow other applications to use the available glyphs, but a straightforward translation from a DVI file would become impossible since \TeX codes are hardwired in such a file.

4 Transforming a \LaTeX document into an SVG document

Note that SVG fonts are not needed if one is content with having the complete EPS figure, including the text, translated into SVG as pure paths descriptors (the text characters are treated merely as graphics paths). Use `pstoedit`'s `-dt` option in that case. This allows for fast rendering but loses the structural information about the graphics, making it more difficult to update. Moreover, when zooming, the quality of the rendering with SVG fonts (left part of Figure 3) is much better than when the \TeX characters are simply translated into SVG paths (right part of Figure 3, where it is seen that the letters of the text are much coarser than at the left).

4.1 Using `pstoedit`

When using `pstoedit` to generate SVG from a PostScript source one needs to include the SVG font instances of the character glyphs referenced. For this we developed an XSLT stylesheet `ins-saxon6.xsl`.²¹ In the following example we

²¹ Versions of the stylesheet are available for Microsoft's `msxsl` (part of their MSDN XML Developer Center <http://msdn.microsoft.com/library/>) and for the Saxon (<http://saxon.sourceforge.net/>) and Xalan (<http://xml.apache.org/xalan-j>) Java XSLT processors.

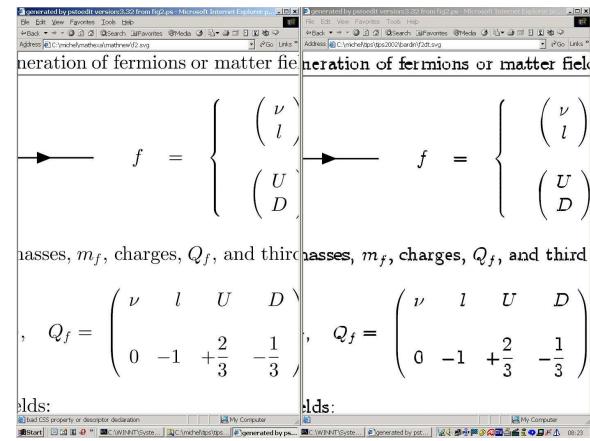


Figure 3: \LaTeX text rendered with SVG fonts or as graphics paths

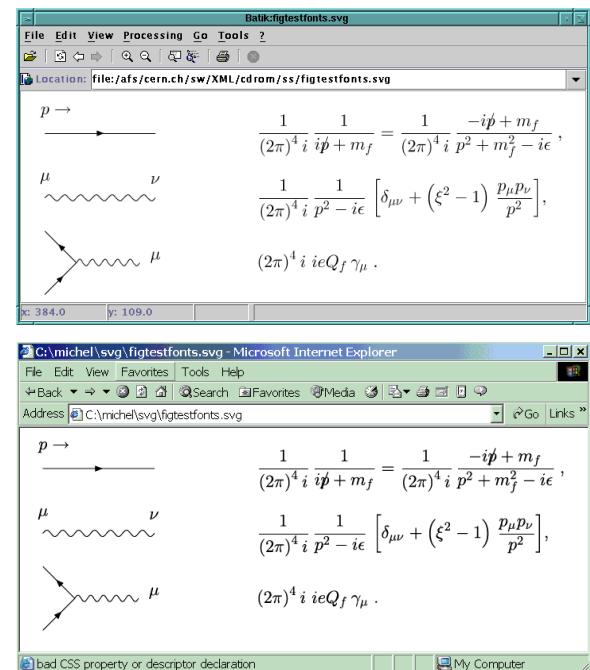


Figure 4: Feynman diagrams and their propagators

transform the PostScript input file `in.ps` into the corresponding SVG instance `out.svg`. Since the latter file does not include the SVG source of the fonts we need to include these by transforming it with the help of an XSLT stylesheet to finally obtain `out1.svg`, which we can view with an SVG capable browser.

```
pstoedit -f svg:-texmode -nfr in.ps out.svg
saxon6.sh out.svg ~/www/svg/ins-saxon6.xsl out1.svg
```

Figure 4 shows a series of Feynman diagrams. The graphics at the left are generated with the help

of PostScript commands interfaced to L^AT_EX, the right part shows their corresponding propagators and are prepared as L^AT_EX math. The source is run through L^AT_EX, turned into PostScript with dvips, before translating it into SVG with pstoedit. After including the SVG fonts with the XSLT script we display the result with Batik (top) and with Microsoft Internet Explorer and the Adobe svgview plugin (bottom).

4.2 Using dvi2svg

We interface the dvi2svg Java library via a small Unix script dvi2svg.sh, whose use is as follows:

```
> dvi2svg.sh
Usage: dvi2svg.sh [options] [DVIFILE]
Options:
 -o [FILENAME] : Specify an output filename prefix. If not
                  set, dvi2svg will take the input filename.
 -d : set the debug mode to on(1)/off(0 default)
```

An example of the use of the dvi2svg program is the translation of the font table of the American Mathematical Society font msbm10 from the DVI format into SVG. In contrast to pstoedit, the dvi2svg program includes itself the SVG font outlines for the needed characters (font sub-setting is used). It is however impossible to deal with non-T_EX material, such as EPS or PDF graphics, in which case pstoedit should be used.

Below we first generate the DVI file using the fonttable utility nfssfont.tex that comes with the L^AT_EX distribution. Then we run dvi2svg.sh on the generated nfssfont.dvi DVI file and obtain the SVG file msbm1.svg.

```
> latex nfssfont
This is TeX, Version 3.14159 (Web2C 7.3.7x)
(/TeXlive/t17/texmf/tex/latex/base/nfssfont.tex
LaTeXe <2001/06/01>
...
*****
* NFSS font test program version <v2.0e>
*
* Follow the instructions
*****
Name of the font to test = msbm10
Now type a test command (\help for help:)
*\table
*\bye
[1]
Output written on nfssfont.dvi (1 page, 5940 bytes).
> dvi2svg.sh nfssfont.dvi -o msbm
DEBUG from converter.DviToSvg => Converting file: nfssfont.dvi
DEBUG from converter.DviToSvg => Writing result to: msbm
DEBUG from converter.DviToSvg => Reader has been created
DEBUG from converter.DviToSvg => Writer has been created
Converting .....FINISHED
> ls -l msbm*.svg
-rw-rw-r-- 1 goossens 161252 Jan 2 10:08 msbm1.svg
```

Figure 5 shows in its bottom part the font table of the font msbm10 as viewed with xdvi, while its top part shows the SVG rendering with Batik of the SVG file msbm1.svg prepared with dvi2svg from the original DVI file, as shown above.

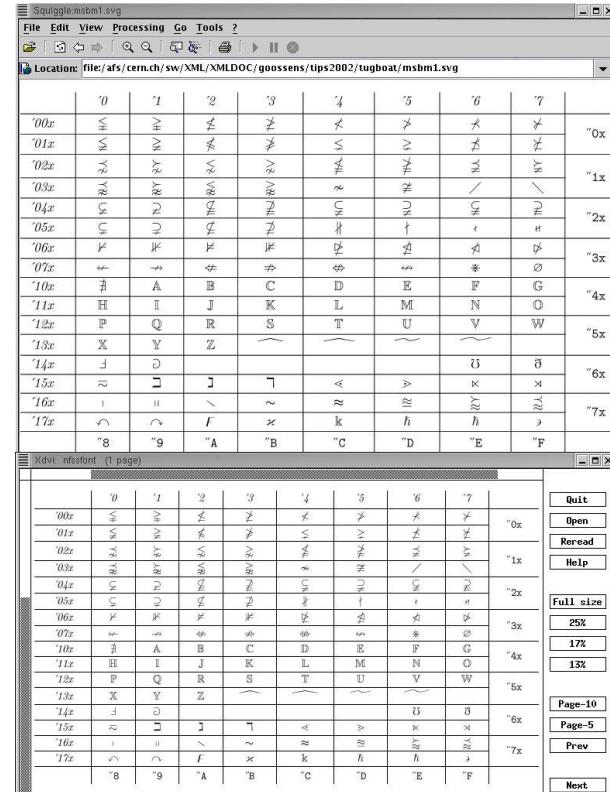


Figure 5: T_EX's msbm10 font table

4.3 More complex examples

Figure 6 displays the result of translating L^AT_EX text and a complex use of its picture environment into SVG in various ways. At the left we show the file as viewed with dvips (PostScript version), in the middle the SVG file created with the pstoedit program from the PostScript output, at the right the SVG file as generated directly from the DVI file with dvi2svg. Both SVG files were displayed with the help of Microsoft's Internet Explorer (with Adobe's svgview plugin). Figure 7 shows part of the SVG source file that was generated by dvi2svg. We shall study its basic structure later (see Section B.2).

We also looked at L^AT_EX sources that use non-standard fonts, such as the MusiX_TE_X and XY-pic packages. We first generated the SVG instances for the Type 1 versions of the needed fonts and then selected a few examples²² that are typical for these packages. For completeness we also include examples of chemistry and algebra. We generated the

²² We took our inspiration from chapters 5 “The XY-pic package” and 7 “Preparing music scores” of *The L^AT_EX Graphics Companion, Illustrating Documents with T_EX and PostScript*, by Michel Goossens, Sebastian Rahtz, and Frank Mittelbach, Addison-Wesley, 1997, ISBN 0201854694.

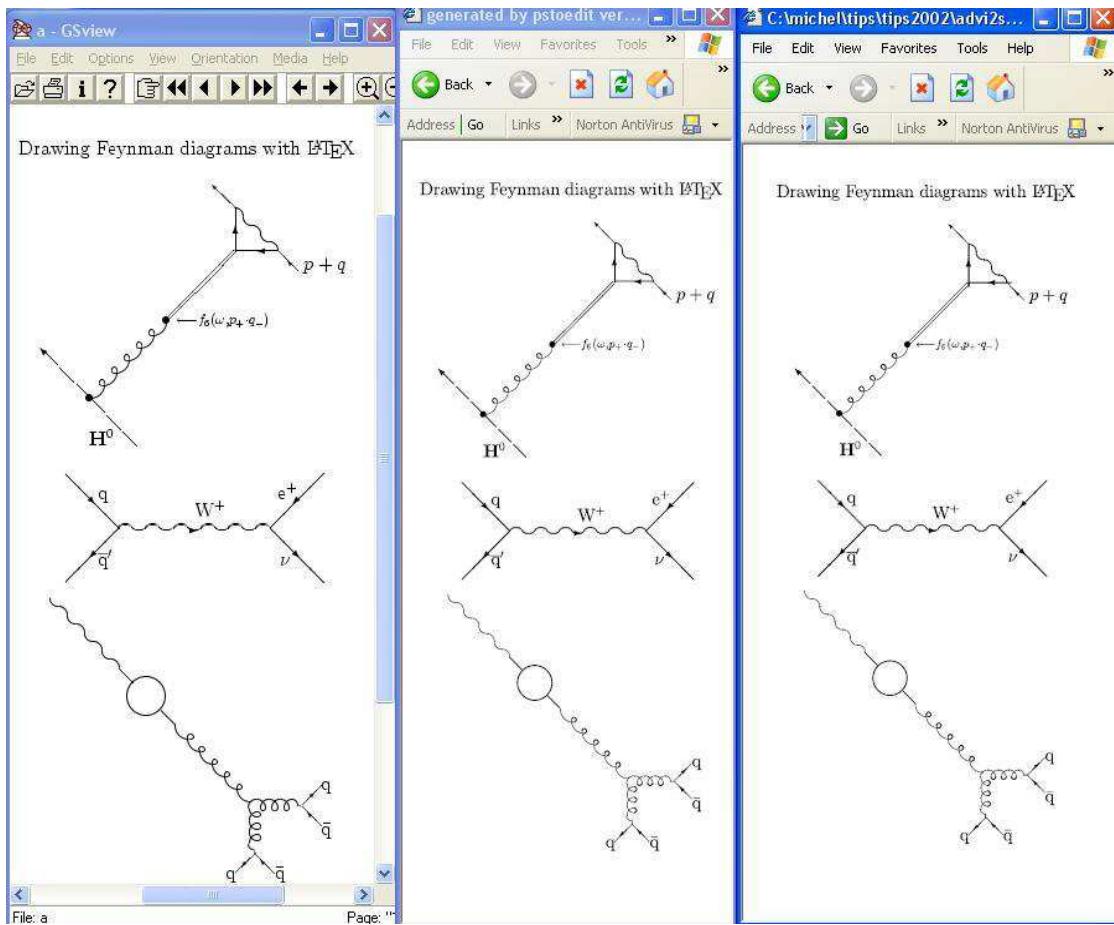


Figure 6: L^AT_EX and its picture environment: PostScript and SVG renderings

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <!--This file was automatically generated by dvi2svg-->
3 <svg width="504" height="800" viewBox="0 0 504 800">
4 <defs>
5   <font id="CMR">
6     <font-face font-family="CMR"/>
7     <glyph unicode="&##57412;" glyph-name="D"
8       horiz-adv-x="763.9">
9       <path style="fill:#000000; fill-rule=evenodd; stroke:none"
10      d="..."/>
11   </glyph>
12   <glyph unicode="&##57458;" glyph-name="r"
13     horiz-adv-x="391.7"&gt;
14     &lt;path style="..." d="..."/&gt;
15   &lt;/glyph&gt;
16   ...
17 &lt;/font&gt;
18 &lt;font id="LCIRCLEW"&gt;
19   &lt;font-face font-family="LCIRCLEW"/&gt;
20   &lt;glyph unicode="&##57352;" glyph-name="a8"
21     horiz-adv-x="1200"&gt;
22     &lt;path style="..." d="..."/&gt;
23   &lt;/glyph&gt;
24   ...
25 &lt;/font&gt;
26   &lt;font id="LCIRCLE"&gt;&lt;font-face font-family="LCIRCLE"/&gt;...&lt;/font&gt;
27   &lt;font id="LINEW"&gt;&lt;font-face font-family="LINEW"/&gt;...&lt;/font&gt;
28   &lt;font id="LINE"&gt;&lt;font-face font-family="LINE"/&gt;...&lt;/font&gt;
29   &lt;font id="CMBX"&gt;&lt;font-face font-family="CMBX"/&gt;...&lt;/font&gt;
30   &lt;font id="CMSY"&gt;&lt;font-face font-family="CMSY"/&gt;...&lt;/font&gt;
31   &lt;font id="CMMI"&gt;&lt;font-face font-family="CMMI"/&gt;...&lt;/font&gt;
32 &lt;/defs&gt;
33 &lt;g&gt;
34   &lt;text fill="black" font-family="CMR" font-size="11.7871"&gt;
35     &lt;tspan y="101.88267" x="...">&lt;...&lt;/tspan&gt;
36   &lt;/text&gt;
37   &lt;rect x="322.78577" y="145.65428" width="0.872714"
38     height="0.26910424" fill="black" stroke="black"
39     stroke-width="0.1"/&gt;
40   ...
41 &lt;/g&gt;
42 &lt;/svg&gt;
</pre>
</div>
<div data-bbox="232 859 763 877" data-label="Caption">
<p><b>Figure 7:</b> L<sup>A</sup>T<sub>E</sub>X and its picture environment: Generated SVG instance</p>
</div>
```

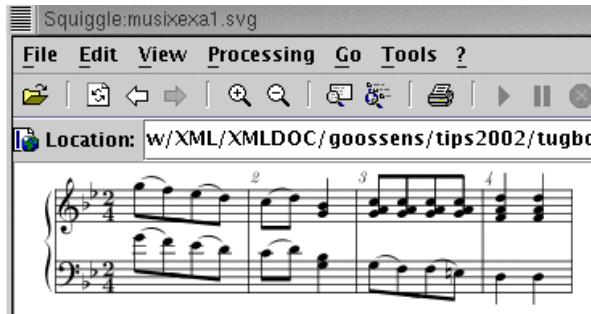


Figure 8: SVG instance of a MusiX \TeX example viewed with Batik Squiggle

SVG from the DVI files as follows (we show only one run, the others are similar).

`dvi2svg.sh xytest.dvi`

```
DEBUG from converter.DviToSvg => Converting file: xytest.dvi
DEBUG from converter.DviToSvg => Writing result to: xytest
DEBUG from converter.DviToSvg => Reader has been created
DEBUG from converter.DviToSvg => Writer has been created
```

Converting

We display the resulting SVG files with Batik's Squiggle SVG viewer. Figure 8 corresponds to example 7–2–5 of *The L^AT_EX Graphics Companion*. It represents a moderately complex part of a music piece typeset with MusiX \TeX . Figure 9 is the structural formula of adonitoxin as typeset with the X $\tilde{\text{\TeX}}$ package (see page 221 of the same book). X $\tilde{\text{\TeX}}$ is an advanced application based on L^AT_EX's `picture` environment. Figure 10 which combines complex mathematical formulae with a graphical representation of a Dynkin diagram generated with the help of the `picture` environment, is based on an entry in a dictionary of Lie superalgebras.²³

Figure 11 shows complex examples from the “Links and knots” Section 5.5.8 of *The L^AT_EX Graphics Companion*. In particular the display is an enlarged view of the bottom parts of examples 5–5–28 and 5–5–29, as well as of examples 5–5–34 and 5–5–35. The display shows that SVG can be nicely scaled for better viewing and is thus a perfect complement for PDF output.

5 Conclusion

We have explained how SVG is a truly scalable two-dimensional XML-based graphics language for the Web. Since graphics representations are at the heart of many scientific, technical and other documents,

²³ The example is taken from the tables in the Appendix of *Dictionary On Lie Algebras And Superalgebras* by Luc Frappat, Antonino Sciarrino, and Paul Sorba, Harcourt Publishers 2000, ISBN 0122653408.

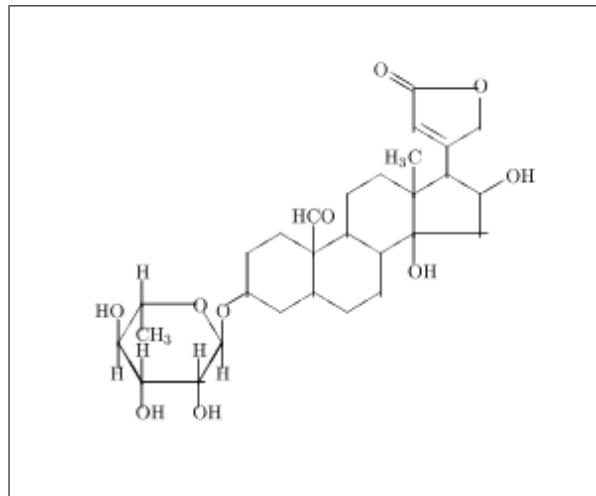


Figure 9: SVG instance of a X $\tilde{\text{\TeX}}$ example viewed with Batik Squiggle

Structure: $\mathcal{G}_{\overline{0}} = sl(m) \oplus sl(n) \oplus U(1)$ and $\mathcal{G}_{\overline{1}} = (\overline{m}, n) \oplus (m, \overline{n})$, type I.

Root system:

$$\Delta = \{\varepsilon_i - \varepsilon_j, \delta_k - \delta_l, \varepsilon_i - \delta_k, \delta_k - \varepsilon_i\}$$

$$\Delta_{\overline{0}} = \{\varepsilon_i - \varepsilon_j, \delta_k - \delta_l\}, \quad \Delta_{\overline{1}} = \{\varepsilon_i - \delta_k, \delta_k - \varepsilon_i\}$$

$$\overline{\Delta}_{\overline{0}} = \Delta_{\overline{0}}, \quad \overline{\Delta}_{\overline{1}} = \Delta_{\overline{1}}$$

where $1 \leq i \neq j \leq m$ and $1 \leq k \neq l \leq n$.

$\dim \Delta_{\overline{0}} = \dim \overline{\Delta}_{\overline{0}} = m^2 + n^2 - m - n + 1$ and $\dim \Delta_{\overline{1}} = \dim \overline{\Delta}_{\overline{1}} = 2mn$.

Distinguished Cartan matrix:

$$\left(\begin{array}{cccc|c|ccc|c} 2 & -1 & 0 & \cdots & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & & & & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 & & & & \\ \vdots & \ddots & \ddots & \ddots & -1 & \ddots & & & \\ 0 & 0 & -1 & 2 & -1 & \ddots & & & \\ \vdots & & \ddots & -1 & 0 & 1 & \ddots & & \vdots \\ \vdots & & & & -1 & 2 & -1 & 0 & 0 \\ & & & & & -1 & \ddots & \ddots & \vdots \\ & & & & & 0 & \ddots & \ddots & 0 \\ & & & & & & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & \cdots & 0 & -1 & 2 \end{array} \right)$$

Distinguished extended Dynkin diagram:

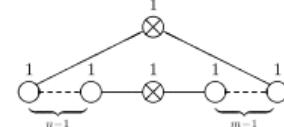


Figure 10: SVG instance of math and a graph viewed with Batik Squiggle

we are convinced that most applications will be able to generate in the foreseeable future SVG output from input data marked up in their specific vocabularies. Similarly, with the help of the Computer Modern family (and other) SVG font sets, which we have explained how to generate, it is now possible to

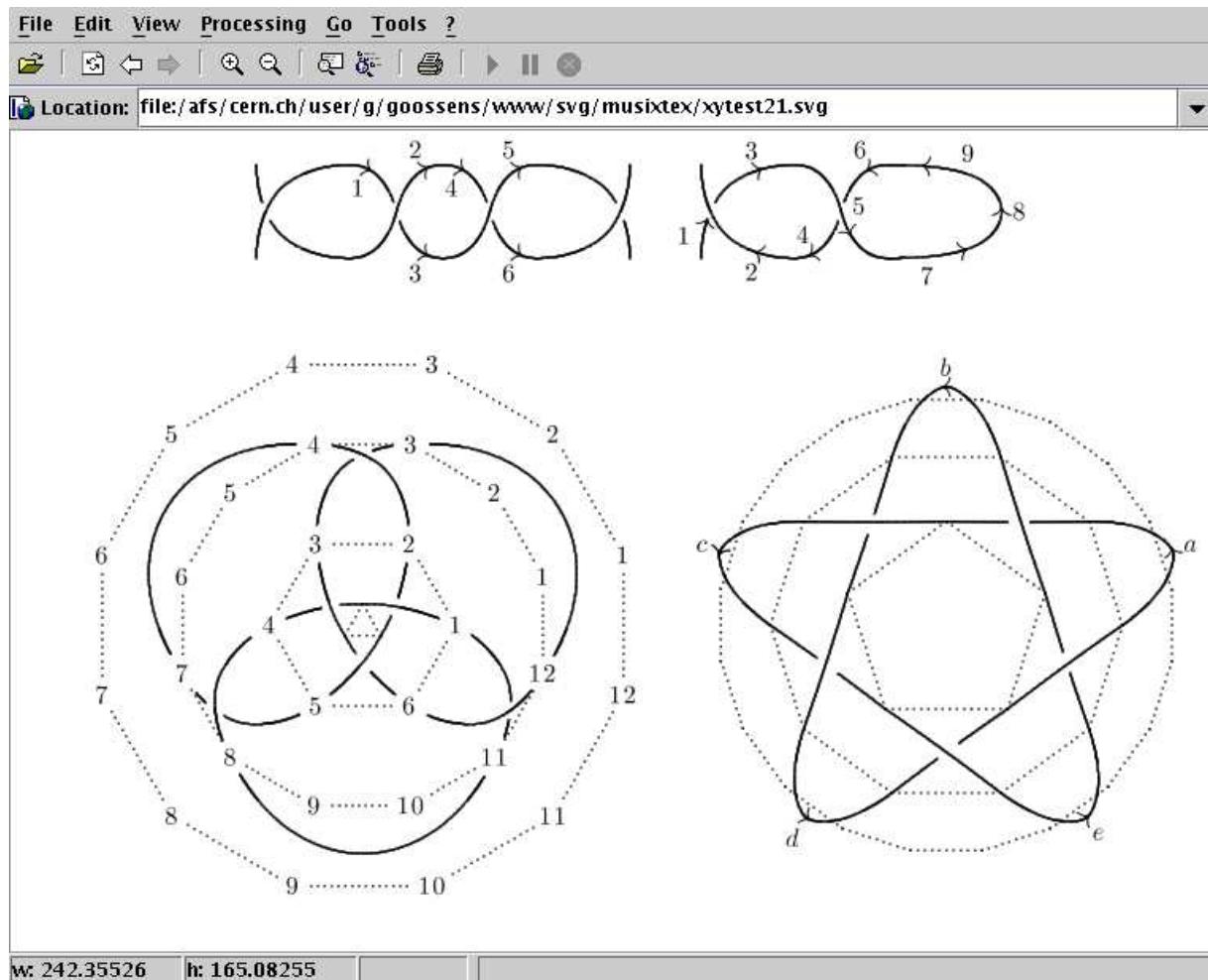


Figure 11: XY-pic example viewed with Batik Squiggle

transform complete L^AT_EX documents into SVG. Although this may prove useful in itself, the existence of PDF and its use inside browsers on the Web makes SVG somewhat redundant in this respect.

However, the more important use of SVG with L^AT_EX will be to translate the graphics images contained in a L^AT_EX document or parts of single pages of L^AT_EX documents from EPS, etc. into SVG so that they can be included in XML instances generated from L^AT_EX sources with the help of L^AT_EX to XML converters, such as **tex4ht**.²⁴ This will become extremely useful once the major browsers are able to handle XML namespaces, making it possible to combine different XML vocabularies. We look forward to the not too distant future when we will be able to generate and edit XHTML (DocBook), MathML

and SVG directly and have the result displayed correctly.²⁵

6 Acknowledgments and distribution

One of us (VS) would like to acknowledge the funding of a fellowship to work at CERN that he received from the “Tools for Innovative Publishing in Science” (TIPS) Project, part of the Information Society Technologies Programme of the 5th Framework of the European Union.²⁶

The latest version of the utilities developed for this project is available as a ZIP file on the Web.²⁷ The fonts also come with the **dvi2svg** distribution.

²⁴ See <http://www.cis.ohio-state.edu/~gurari/TeX4ht> and references therein.

²⁵ See <http://www.w3.org/TR/XHTMLplusMathMLplusSVG/> for some work that is been done in this area.

²⁶ See <http://www.cordis.lu/ist/ist-fp5.html>.

²⁷ See <http://home.cern.ch/goossens/svgfonts.html>.

A SVG graphics path commands

Paths (defined in SVG using the `path` element) specify the geometry of the outline of an object. Path operators can set the current point (`moveto`), draw a straight line (`lineto`), draw a cubic Bézier curve (`curveto`), and close the current shape (`closepath`). Details on a subset of the SVG path commands that are used in SVG fonts follow. Command names in lowercase are for relative coordinates, uppercase names for absolute coordinates.

[m|M] (x y)+

Start a new sub-path at the given (x, y) coordinate. A relative `moveto` (`m`) appearing as the first element of a path is treated as a pair of absolute coordinates. If a `moveto` is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit `lineto` commands.

[z|Z]

Closes the current subpath by drawing a straight line from the current point to the current subpath's initial point.²⁸

The various `lineto` commands draw straight lines from the current point to a new point:

[l|L] (x y)+

Draws a line from the current point to the given (x, y) coordinate which becomes the new current point. A number of coordinate pairs may be specified to draw a polyline. At the end of the command, the new current point is set to the final coordinate provided.

[h|H] x+

Draws a horizontal line from the current point (cpx, cpy) to (x, cpy) , which becomes the new current point.

[v|V] y+

Draws a vertical line from the current point (cpx, cpy) to (cpx, y) , which becomes the new current point.

There are three groups of commands to draw curves. Here we only look at one of the cubic Bézier commands, since it is used in the translation of the Type 1 fonts. Further information on the other commands are in the SVG Specification.

[c|C] (x1 y1 x2 y2 x y)+

Draws a cubic Bézier curve from the current point to (x, y) using $(x1, y1)$ as the control point at the beginning of the curve and $(x2, y2)$

²⁸ At the end of the command, the new current point is set to the initial point of the current subpath, so that if a `closepath` is followed immediately by any other command, then the next subpath and the current subpath share their same initial point.

as the control point at the end of the curve. Multiple sets of coordinates may be specified to draw a polybézier. The new current point is set to the final (x, y) coordinate pair used in the polybézier.

B SVG Fonts

Graphics designers creating SVG content using arbitrary fonts need to be sure that the same graphical result will be displayed when the content is viewed by all end users, even those who do not have the necessary fonts installed on their computers. Therefore, to guarantee reliable font delivery the SVG Specification defines a common “SVG font” format that all conforming SVG viewers must support.

B.1 Overview

SVG fonts contain unhinted font outlines. Because of this, on many implementations there will be limitations regarding the quality and legibility of text in small font sizes. For increased quality and legibility in small font sizes, or for faster delivery of Web pages (SVG fonts are expressed using SVG elements and attributes, so that they can be quite verbose compared to other formats) alternate font technology might be considered on some systems.²⁹

SVG fonts and their associated glyphs do not specify bounding box information, so that it is up to the applications to calculate bounding box and overhang based on an analysis of the graphics elements contained within the glyph outlines.

B.2 The font element

An *SVG font* is defined using a `font` element.³⁰ The characteristics and attributes of SVG fonts follow closely the font model of the *Cascading Style Sheets (CSS) level 2 Specification*.³¹

²⁹ The authors of the SVG Specification have realized that the absence of a hinting mechanism in the font format of current SVG 1.1 is a drawback. Indeed, Web developers embed fonts in other formats in their SVG documents in situations where the available resolution is insufficient for adequate rendering in native SVG. Therefore, SVG 1.2 (<http://www.w3.org/TR/SVG12>) plans to add hinting as an optional feature for SVG fonts, thus offering Web authors the choice of a pure SVG solution. The adopted approach is likely to be based on a free variant of PostScript Type 1 hinting.

³⁰ See <http://www.w3.org/TR/SVG/fonts.html>.

³¹ See <http://www.w3.org/TR/REC-CSS2/fonts.html>. In that document font metrics are expressed in units that are relative to an abstract square whose height is the intended distance between lines of type in the same type size. This square is called the *em square* and it is the design grid on which the glyph outlines are defined. The value of the `units-per-em` attribute on the `font` element specifies how many units the em square is divided into. Common values are 1000 (Type 1) and 2048 (TrueType or OpenType).

An **font** element can contain the following elements: **font-face**³² (provides further typographic information about the font, including the name of the font), **hkern** and **vkern** (kerning information between Unicode characters), **missing-glyph** (defines the representation to be used for all Unicode characters that have no explicit **glyph** element defining their outline in the present font), and finally **glyph**.

The **glyph** element defines the graphics for a given glyph. The coordinate system for the glyph is defined by the various attributes in the **font** element. The graphics that make up the **glyph** can be either a single *path data* specification within the **d** attribute (see below) or arbitrary SVG as content within the **glyph** element.

Important attributes of the **glyph** element are described below.

unicode = "<string>"

If a single character is provided, then this **glyph** corresponds to the given Unicode character. If multiple characters are provided (e.g., for ligatures) then this **glyph** corresponds to the given sequence of Unicode characters. For example see line 1 of Table 2 and lines 7, 12, and 20 of Figure 7.

glyph-name = <name> [,<name>]*

A **glyph name** should be unique within a font. **Glyph names** are used when Unicode character numbers do not provide sufficient information to access the correct **glyph** (e.g., when there are multiple **glyphs** per Unicode character). **Glyph names** are referenced in *kerning* definitions. For example see line 1 of Table 2 and lines 7, 12, and 20 of Figure 7.

d = "path data"

Definition of the outline of a **glyph**. Uses the same syntax as the **d** attribute on a **path** element, which is often used instead. For example see lines 4–27 of Table 2 or lines 10, 14, 22 of Figure 7.

horiz-adv-x = "<number>"

The default horizontal advance after rendering a **glyph** in horizontal orientation. **Glyph widths** must be non-negative, even if the **glyph** is rendered right-to-left, as in Hebrew and Arabic scripts. An attribute **horiz-adv-y** exists for specifying the vertical advance for **glyphs** rendered in vertical orientation. For example see line 2 of Table 2 and lines 8, 13, and 21 of Figure 7.

³² Similar to CSS2's **@font-face** font descriptor, see <http://www.w3.org/TR/REC-CSS2/fonts.html>.

For increasing portability it is advisable to embed all SVG fonts that are referenced inside an SVG document. As an example, Figure 7 shows how all the fonts needed to render the given SVG graphics image are first included (inside a **defs** element, lines 4 to 32) and later referenced (e.g., line 34 calls for a character of font **CMR** whose definition is on lines 5–17).

On the other hand, it is also possible, e.g., for convenience, to save SVG font sources in external files and reference characters in these fonts via CSS style directives from inside SVG images. In this case one must, however, make sure that the needed fonts are installed on the client's system or are shipped together with the referencing SVG file to the client site.

- ◊ Michel Goossens
IT Division, CERN
CH1211 Geneva 23, Switzerland
michel.goossens@cern.ch
- ◊ Vesa Sivunen
ETT Division, CERN
CH1211 Geneva 23, Switzerland
vesa.sivunen@cern.ch

mimeTeX announcement

John Forkosh

Introduction

This short note announces the availability of **mimeTeX**, a small GPL'ed program that facilitates the preparation of HTML documents containing math. **mimeTeX** parses L^AT_EX-like math expressions, emitting either MIME xbitmaps or GIF images of them, which can be used in HTML documents e.g.,

```

```

This allows you to embed math directly in HTML, reducing the need for external GIF images, and making your HTML documents more readable and easily maintained.

You can see detailed documentation and examples online at <http://www.forkosh.com/mimetex.html>, and the entire package can be downloaded from [/tex-archive/support/mimetex/mimetex.zip](http://tex-archive/support/mimetex/mimetex.zip) at any CTAN mirror.

mimeTeX isn't primarily meant for `latex2html`-like tasks where you're maintaining native L^AT_EX documents that are later redistributed in several formats, including HTML. Rather, **mimeTeX** is primarily meant to help maintain native HTML documents containing math. In this sense, it's a kind of "lightweight" alternative to MathML, with the advantage that **mimeTeX** preserves easy-to-use L^AT_EX syntax. And **mimeTeX** works with any graphical browser.

mimeTeX's objectives

Widespread use of MathML by HTML/XML authors will eventually begin to dilute the population of L^AT_EX-aware users, muddying L^AT_EX's future. L^AT_EX is more than "T_EX The Program"; L^AT_EX is its syntax. Knuth produced a test suite that validates any program claiming to be T_EX, so no one version of the code is crucial. It's the syntax that's crucial. L^AT_EX will survive so long as a significant user population continues to use this syntax.

MathML poses a threat to the future of L^AT_EX's syntax in the large and growing HTML/XML market, so it's useful and important to provide some L^AT_EX-compliant alternative. **mimeTeX** is meant to be a prototype alternative. It's probably too small and kludgy for a final solution. But it demonstrates feasibility, and is full-featured enough to measure potential interest in L^AT_EX-compliant alternatives to MathML.

Such alternatives provide a choice to new users, who will hopefully conclude that L^AT_EX is the easier and more intuitive syntax. And old users can continue using L^AT_EX syntax when they have to prepare native HTML/XML documents, i.e., when it's not adequate to run `latex2html` against native L^AT_EX documents.

Similar tools

Other non-MathML solutions besides **mimeTeX** that embed L^AT_EX-like math into HTML are discussed in the T_EX FAQ. Two that you might want to look at are `textogif` at <http://www.fourmilab.ch/webtools/textogif/textogif.html> and `gladTeX` at <http://www.math.uio.no/~martingu/gladtex>. Both require separate setup procedures that use T_EX to help generate external GIF (or PNG) images of your equations, which are later included in your HTML document as it's being rendered.

mimeTeX, as far as I know, is the only such non-MathML package that has its own built-in parser and rendering engine, entirely independent of T_EX, and therefore requires no setup procedure or external images whatsoever. It renders realtime, on-the-fly

images directly from your L^AT_EX math embedded in HTML documents. This makes your HTML source documents more readable and easily maintained. `textogif`, `gladTeX`, or similar tools may be modifiable to work as easily, or **mimeTeX**'s ease-of-use features may not prove compelling. In any case, **mimeTeX** becomes one more available tool in your toolbox.

◊ John Forkosh
285 Stegman Parkway #309
Jersey City, NJ 07305
USA
john@forkosh.com
<http://www.forkosh.com>

Fonts

Making outline fonts from bitmap images

Karl Berry

Abstract

Tools for creating outline fonts from bitmaps have matured significantly in past years. Our purpose here is to describe those tools and how they fit together in a *TEX* context, in a practical way.

1 Introduction

I recently had occasion to create outline fonts (PostScript Type 1 or TrueType; see [24]) from a scanned bitmap image of a type specimen. I was pleasantly surprised to find that the tools for doing this conversion produce considerably better results than the last time I worked in this area. This note describes one procedure for putting the programs together, culminating in using the result from *TEX*.

Happily, all of the programs mentioned here can be compiled and installed with only minor variations on the standard procedure [18] first specified for the GNU project:

```
configure && make && make install
```

Web addresses for the programs are given in the references.

We should at least touch on legal questions [4], although a thorough discussion is far beyond the scope of this paper—or my knowledge, for that

matter. My understanding is that font designs, as opposed to font programs, are still not copyrightable in the United States (a few have been patented, notably Lucida), but that designs are protected in most European countries. As a result, in today's world of widespread file sharing, especially in the TEX community, it would be unwise to attempt to create or distribute fonts for any design created after approximately the early 1900's, without specific knowledge for a specific design.

On the other hand, I did receive legal advice (from the Free Software Foundation's lawyer) that scanning old type specimen images, even when they are embedded in a book still under copyright, is defensible. Not any of the text or illustrations prepared specifically for the book, of course, but actual old specimens may be copied.

2 Scanned image to bitmap font: `imageto`

For our purposes, we will start with a single black and white image of a font specimen of a Baskerville type at a fairly large size (24 pt), scanned at a fairly high resolution (1200 dpi). The image includes the upper and lowercase alphabets, digits, and other principal characters. The first task is to extract these characters from the image into a bitmap font.

Not coincidentally, one of the programs in the GNU font utilities [3], written a decade or so ago by Kathryn Hargreaves and myself, does precisely this. This program is called `imageto`. There may be other programs to accomplish the same task, but since I knew about this one (for obvious reasons), I just used it.

The output from the scanner (a Xerox 9700) is in an unusual image format that can't be read directly by any modern program. (The scanning was also done a decade ago.) So, to see the image I was working with, I converted it to Encapsulated PostScript (EPSF [24]) and viewed it with `gv` [13], at its smallest scale factor (0.1):

```
imageto --epsf gbvr
gv gbvr.eps
```

Here's the resulting picture of the starting image (clipped to approximately the left half due to the small *TUGboat* column width), so we can see what we're dealing with:

The font name `gbvr`, by the way, stands for GNU Baskerville roman, according to the Fontname scheme [2]. (This whole project started under the aegis of GNU [19].)

`imageto` considers the input image as a series of ‘image rows’. Each image row consists of all the scanlines between a nonblank scanline and the next entirely blank scanline. (A ‘scanline’ is a single horizontal row of pixels in the image.) Within each image row, `imageto` looks top-to-bottom, left-to-right, for ‘bounding boxes’: closed contours, i.e., an area whose edge you can trace with a pencil without lifting it. For example, an ‘i’ has two bounding boxes, while an ‘a’ has one.

In practice, scanned images have plenty of imperfections; for instance, a small printing blotch is seen as a bounding box which we have to ignore. Baselines jump up and down due to the printing process, as well as the natural baseline adjustments for characters with descenders or o-corrections (curved characters such as ‘o’ whose bottom point is slightly below the baseline). So we have to describe all of these special cases.

To extract characters from the bitmap, it's necessary to supply all of this descriptive information to `imageto` in a simple line-oriented text file, called an `ifi` file (image font information). The details of this file's syntax aren't important here—the full manual (and source code) for Fontutils are available at the url given in the references.

For the sake of example, the final command line to process this image ended up being:

```
imageto --designsize=24 \
--encoding=8r \
--baselines=72,59,58 \
--print-guidelines \
--print-clean-info \
gbvr
mv gbvr24.1200gf gbvr.1200gf
```

The result is a bitmap font in GF format [9] (same as METAFONT). So, from `gbvr.img`, we now have `gbvr.1200gf`. We explicitly remove the 24 in the filename because we want to make an outline font, named without a design size.

3 Bitmap font to outlines: `autotrace`

The best program I know of to fit outlines (i.e., Bézier curves) to bitmaps is `autotrace` [23]. (I found an alternative program `ttf2pt1` [1], but it did not seem as well developed). Some fairly intense mathematics is involved in doing the fitting [17]; fortunately, we don't need to go into that in order to use the program effectively.

The main barrier to using `autotrace` to convert fonts is that it reads images (such as PBM files [14]), and writes EPSF (among other formats); it has no knowledge of font formats. So our basic strategy is:

1. Convert each character in the bitmap font to an image in PBM format.
2. Run `autotrace` on that image.
3. Convert the PostScript output, which uses standard graphics operators such as `rmoveto` and `rrcurveto`, to Type 1 opcodes.
4. Reassemble the characters into a font.

This procedure is implemented by `mftrace` [11], a Python [22] program which pulls the pieces together: it uses `gf2pbm` [12] to convert individual characters from the GF font to bitmap images; calls `autotrace` with assorted options to do the fitting; and finally uses `t1asm` from `t1utils` [7] to assemble the output into a font again.

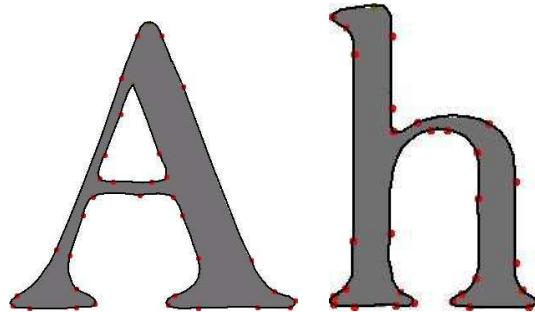
`mftrace` requires an encoding file to run, and since it does not do path searching, the encoding file must be present in the current directory. The default encoding is `tex256.enc`, and can be changed with the `-e` option. (If you don't happen to have `tex256.enc` on your system, it's available at <http://tug.org/fontname/tex256.enc>. It is another name for the T1 (Cork) encoding.)

Type 1 fonts have two equivalent formats: pfa (printer font ASCII), which uses only normal plain text characters; and pfb (printer font binary), which is partly binary. As you might expect, pfb files are noticeably smaller, since the font shapes can be compressed more when all eight bits can be used. `TEX` and friends are happy with either one, so we might as well use pfb.¹

Here is a first approximation to our command line:

```
mftrace --pfb \
--gffile=gbvr.1200gf gbvr
```

The result is `gbvr.pfb`. Here are two of the resulting outline characters showing the main control points:



There is an alternative program `textrace` [20], which seems equally worthy. I worked with `mftrace` only because it was easier for me to install and understand. Each program has its own drawbacks and benefits.

Historical aside: I was happy to see that `autotrace` is partially based on `limn`, another of the old Fontutils programs; so that work wasn't entirely wasted. It does a vastly better job than `limn` ever did, which I am even happier to see!

4 Testing the new font from `TEX`

The above does the real work of converting bitmaps to outlines. Now, to use the result in `TEX`, we have many configuration details to work out.

4.1 Scaling: `mftrace`

First, we need metrics to go along with the outlines, which `mftrace` will generate as an afm file (Adobe Font Metrics [24]), if we specify the `--afm` option.

We also have to convert between different coordinate systems. When we create a Type 1 font, the so-called 'character coordinate space' uses a 1000 to 1 scaling matrix. That is, 1000 character space units transform into one user space unit (one PostScript point, usually). Put another way, Type 1 expects a resolution of 1000 pixels in the design size, and the PostScript design size is simply 1. This is a much higher resolution than our scanned images have.

It's easiest to explain the scaling factor by looking at a concrete example. Our example image was scanned at a resolution of 1200 dpi, which comes to about 16.6 pixels per point (1200/72.27). Our design size is 24 pt. Therefore, we have about 399 pixels per design size (16.6 * 24). We give this value to the `mftrace` as the magnification (same concept of magnification as in `TEX`).

`mftrace` will then scale all the numbers in the outlines by $1000/399 = 2.506$. For example, the image of our capital 'A' is 275 pixels wide. This becomes about 690 in character space coordinates. As a check, the device-independent width in `TEX` terms turns out to be 6.9 on a 10 pt designsize; yay.

¹ Warning: I found out to my sorrow that it does not work to directly edit the contents of a pfb file in any way, including the header comments; it becomes internally inconsistent and `dvips` will complain about a 'non-MSDOS header'.

Bottom line, the magic number is the image design size multiplied by the image resolution in pixels per point. Here's the resulting `mftrace` command line (this is the real one, no approximation):

```
mftrace --pfb \
--afm --magnification=399 \
--gffile=gbvr.1200gf gbvr
```

We now have two files: `gbvr.pfb` and `gbvr.afm`.

4.2 Metrics: afm2tfm

Our next job is to convert `gbvr.afm` into `TEX` font metric files. The easiest way I know of to do this is to use `afm2tfm`'s `-T` option, which lets us get away without using virtual fonts [26]:

```
afm2tfm gbvr.afm -v gbvr.vpl \
-T tex256.enc >gbvr.xmap
pltotf gbvr.vpl gbvr.tfm
```

`pltotf` will issue warnings about unknown `VTITLE` and `MAPFONT` properties, but no harm is done. We don't need all the `TEX` virtual font machinery [5], since we're not actually combining multiple fonts, just reencoding a single font.

4.3 Running `TEX` and `dvips`

Ok, let's run `TEX` (`testfont.tex` is a standard file from Knuth):

```
tex testfont
...
Name of the font to test = gbvr at 24pt
Now type a test command...
*\text\bye
...
Output written on testfont.dvi...
Transcript written on testfont.log.
```

We're almost ready to look at some output. Our last preliminary step is to specify downloading `gbvr.pfb` when the font is used. For `dvips` [15], this is done in a one-line `.map` file. `afm2tfm` gave us the initial line, we just append the download instruction:

```
sed -e 's/$/ <gbvr.pfb/' gbvr.xmap \
>gbvr.map
```

Finally, we tell `dvips` [15] to use that map file and process the document:

```
dvips -u +gbvr.map testfont.dvi -o
```

The output is `testfont.ps`:

On November 14, 1885, Senator & Mrs. Leland Stanford called together at their San Francisco mansion the

There are some artifacts in that image due to the conversion process from the screen capture. My

apologies, but the main point is the process, after all, not the particular image we used for an example.

The most obvious other problem is that there is no letter spacing. Some methods for addressing that are mentioned in the next section.

5 Final outline output: `frontline`, `pfaedit`

We've gone through the process of making a new outline font and typesetting with it in `TEX`. Now comes the truly hard part: actually making the font as good as it can be. Although `autotrace` does a very respectable job with its default settings, it's inevitable that hand editing of the outlines will be required for best results.

One method for doing this is to change the (numerous) parameters to `autotrace` itself. This can be done from the command line (use `--help` to get a list of options). In addition, a graphical front-end to `autotrace` named `frontline` exists to make experimenting with the option setting easier; it is available from the `autotrace` home page [23].

The other method is to use an outline font editor; the best one I know of is `pfaedit` [25]. As well as straightforward outline editing, `pfaedit` has numerous other significant features:

- Bitmap editing (supports GF and PK [16] formats).
- TrueType output. The option `--truetype` to `mftrace` will call `pfaedit` to get TrueType output, if that's desired.
- Metrics editing: getting the character spacing is as important to the final outcome as the character shapes [8, 21]. `pfaedit` can supply initial side bearings and kerns via the `Auto Width` and `Auto Kern` options on the `Metrics` menu. This is nice, since scanned images generally lack any useful side bearing specifications. (Alternatively, the `Fontutils` program `charspace` is a non-interactive way of preparing initial side bearings.)
- Autohinting.

As it turns out, the `mftrace --afm` option that we used above also implies `--simplify`, which runs the font through `pfaedit` in order to simplify and autohint the outlines. Thus, no additional options are needed to take advantage of those features.

Additional files and procedures are needed to use new fonts with `LETEX` [6, 10]. Those articles also describe creating oblique, small caps, and other variants.

Happy fontmaking!

References

- [1] Sergey Babkin. ttf2pt1.
<http://ttf2pt1.sourceforge.net>.
- [2] Karl Berry. Fontname: Filenames for TeX fonts. <http://tug.org/fontname>.
- [3] Karl Berry and Kathryn Hargreaves. GNU fontutils. <http://www.gnu.org/software/fontutils>.
- [4] Charles Bigelow. Notes on typeface protection. *TUGboat*, 7(3):146–151, October 1986.
- [5] Robin Fairbairns. Virtual fonts.
<http://www.tex.ac.uk/cgi-bin/texfaq2html?label=virtualfonts>.
- [6] Peter Flynn. Installing PostScript fonts.
<http://www.silmariel.ie/downloads/documents/installpsfonts.pdf>.
- [7] I. Lee Hetherington and Eddie Kohler. Type 1 utilities (t1utils).
<http://www.lcdf.org/~eddietwo/type>.
- [8] David Kindersley. *Optical Letter Spacing for New Printing Systems*. Wynkyn de Worde Society, distributed by Lund Humphries Publishers Ltd., 26 Litchfield St. London WC2, 1976.
- [9] Donald E. Knuth. GF (generic font) format.
<http://www.ctan.org/tex-archive/systems/knuth/mfware/gftype.web> (among other programs).
- [10] Philipp Lehman. The font installation guide.
<http://www.ctan.org/tex-archive/info/Type1fonts/fontinstallationguide.pdf>.
- [11] Han-Wen Nienhuys. mftrace.
<http://www.cs.uu.nl/~hanwen/mftrace>.
- [12] Han-Wen Nienhuys and Paul Vojta. gf2pbm.
<http://www.cs.uu.nl/~hanwen/mftrace>.
- [13] Johannes Plass. GV: a PostScript and PDF viewer. <http://wwwhep.physik.uni-mainz.de/~plass/gv>.
- [14] Jef Poskanzer and Bryan Henderson et al. Netpbm. <http://netpbm.sourceforge.net>.
- [15] Tomas Rokicki. Dvips. <http://www.ctan.org/tex-archive/dviware/dvips>.
- [16] Tomas Rokicki. PK (packed font) format.
<http://www.ctan.org/tex-archive/systems/knuth/mfware/pktype.web> (among other programs).
- [17] Philip J. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, University of Washington, 1988.
http://autotrace.sourceforge.net/Interactive_Curve_Design.ps.gz.
- [18] Richard M. Stallman. GNU coding standards.
http://www.gnu.org/prep/standards_48.html. Node: Managing Releases.
- [19] Richard M. Stallman. Project GNU (GNU's Not Unix). <http://www.gnu.org>.
- [20] Péter Szabó. textrace.
<http://textrace.sourceforge.net>.
- [21] Walter Tracy. *Letters of Credit*. David R. Godine, Publisher, Boston, MA, USA, 1986.
- [22] Guido van Rossum. Python.
<http://python.org>.
- [23] Martin Weber. Autotrace.
<http://autotrace.sourceforge.net>.
- [24] George Williams. Font file formats.
<http://pfaedit.sourceforge.net/index.html#Formats>. This has contains links to PostScript Type 1 and AFM documents, and both the Apple and Microsoft TrueType and OpenType standards documents, among many others.
- [25] George Williams. Pfaedit.
<http://pfaedit.sourceforge.net>.
- [26] Y&Y. Single TFM file for Type 1 Fonts.
<http://www.yandy.com/maketfm.htm>.

◊ Karl Berry
685 Larry Ave. N
Keizer, OR 97303
USA
karl@freefriends.org
<http://freefriends.org/~karl/>

Software & Tools

Size reduction of chemical structural formulas in X^MT_EX (Version 3.00)

Shinsaku Fujita* and Nobuya Tanaka

1 Introduction

The X^MT_EX system (Version 2.00) [1], which we released as an implementation of the X^M Notation [2] and the X^M Markup Language [3], has provided a convenient method for drawing complicated structural formulas. The X^MT_EX system has been designed to assure maximal portability within the scope of L^AT_EX/L^AT_EX 2_ε [4, 5]. The version 2.00

* To whom correspondence should be addressed.

has, however, suffered from a drawback that the size reduction of structural formulas has not been permitted. This has come from the fact that the $\text{\texttt{X}}\tilde{\text{\texttt{M}}}\text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$ system has depended on the $\text{\texttt{L}}\tilde{\text{\texttt{A}}} \text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$ picture environment that has been incapable of drawing short bonds (lines). Although the $\text{\texttt{epic}}$ system [6] has been used to draw short lines so as to maintain such portability, it has occasionally given a split line. For example, the commands of the $\text{\texttt{epic}}$ system,

```
\drawline(0,0)(171,103) and  
\drawline(0,0)(171,-103),
```

are necessary to draw a benzene ring but give the following split lines:



when we encounter the worst-case situation (e.g., under $\text{\texttt{unitlength}}=0.08\text{pt}$). If we lay stress on the portability of a drawing system [7], one of the most promising ways is to rely on the $\text{\texttt{epic}}$ system after we analyze and revise the mechanism of giving split lines. Hence, the aim of this paper is to show how the $\text{\texttt{X}}\tilde{\text{\texttt{M}}}\text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$ system (Version 3.00) [8] provides a method for permitting the size reduction of structural formulas within the scope of the $\text{\texttt{L}}\tilde{\text{\texttt{A}}} \text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$ picture environment and the $\text{\texttt{epic}}$ system.

2 Basic functions for size reduction

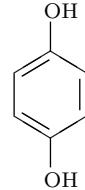
2.1 sizeredc package

The command $\text{\texttt{lineslope}}$ of $\text{\texttt{epic}}$ has been used to convert the command $\text{\texttt{drawline}}$ of $\text{\texttt{epic}}$ into the command $\text{\texttt{line}}$ of the $\text{\texttt{L}}\tilde{\text{\texttt{A}}} \text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$ picture environment. In the process of obtaining the slope of a line, the command $\text{\texttt{linslope}}$ has occasionally provided a rounding error, which has been found to cause such split lines as described above. A simple remedy for this phenomenon has been given in the $\text{\texttt{sizeredc}}$ package (file name: $\text{\texttt{sizeredc.sty}}$) distributed as a part of the present version of $\text{\texttt{X}}\tilde{\text{\texttt{M}}}\text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$. According to this remedy, the drawing mechanism of the $\text{\texttt{X}}\tilde{\text{\texttt{M}}}\text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$ system can be safely switched into the mechanism of $\text{\texttt{epic}}$, if $\text{\texttt{unitlength}}$ is set to be smaller than 0.1pt . Note that the unit length of the $\text{\texttt{X}}\tilde{\text{\texttt{M}}}\text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$ system is stored by the command $\text{\texttt{unitlength}}$, the standard value of which is 0.1pt .

2.2 Changing unit lengths

The unit length of $\text{\texttt{X}}\tilde{\text{\texttt{M}}}\text{\texttt{T}}\tilde{\text{\texttt{E}}} \text{\texttt{X}}$ can be changed by the command $\text{\texttt{changeunitlength}}$, which is defined in the $\text{\texttt{sizeredc}}$ package. As shown in the following code, the setting by $\text{\texttt{changeunitlength}}$ can be done in the preamble of a document if the value is used in the whole document.

```
\documentclass{article}  
\usepackage{carom}  
\usepackage{sizeredc}  
\changeunitlength{0.08pt}  
\begin{document}  
\footnotesize  
\bzdrv{1==OH;4==OH}  
\end{document}
```



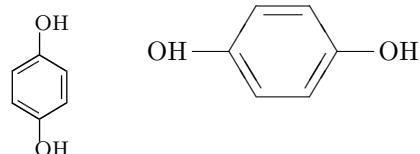
The font size of substituents can be changed by such a command as $\text{\texttt{footnotesize}}$, as shown in the above formula. This should be compared with the counterpart drawn with the standard unit length (0.1pt) and the font size of $\text{\texttt{normalsize}}$.



The command $\text{\texttt{changeunitlength}}$ can be declared at anywhere in a document; the setting of the command is effective after the declaration, until an alternative declaration is carried out. The grouping technique can be used to limit the effect of the setting within a pair of braces. For example, the codes represented by

```
{%grouping by braces  
\changeunitlength{0.06pt}  
\footnotesize  
\bzdrv{1==OH;4==OH}  
\qquad \bzdrv{1==OH;4==OH}
```

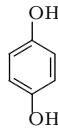
produce the following size-reduced formula and the corresponding formula of the standard dimension:



The command $\text{\texttt{changeunitlength}}$ sets a unit length given as an argument and declares a flag represented by $\text{\texttt{sizereductiontrue}}$ if the argument is less than 0.1pt . The flag is used to substitute the $\text{\texttt{drawline}}$ command of $\text{\texttt{epic}}$ for the $\text{\texttt{line}}$ command

of L^AT_EX 2 _{ε} . Hence, the following setting is equivalent to the setting derived from the declaration command `\changeunitlength{0.05pt}`.

```
{%
\scriptsize
\unitlength=0.05pt
\sizereductiontrue
\bzdrv{1==OH;4==OH}
}
```



3 Examples of size reduction

3.1 Size reduction of carbocycles

When `\sizereductiontrue` is not specified (i.e., `\sizereductionfalse`), the original picture environment of L^AT_EX 2 _{ε} works. Table 1 shows the comparison between cases with and without the use of `sizeredc.sty`, which simulates the difference between X^MT_EX Version 3.00 and Version 2.00.

Without using the `sizeredc` package, X^MT_EX commands such as

```
{\unitlength=0.07pt \bzdrv{}}
{\unitlength=0.06pt \bzdrv{}}
```

give incomplete formulas of benzene that have no inner double bonds (slanted lines), as found in the left column of Table 1. The disappearance of the inner bonds are in agreement with the original specification of the L^AT_EX picture environment. In fact, the `\line` command with slopes (5, 3) and (5, -3) cannot draw extremely short lines, although it is promised to draw longer lines under usual conditions (e.g., `\unitlength=0.1pt` or `0.08pt` without using the `sizeredc` package). By using the commands of `sizeredc` such as

```
{\changeunitlength{0.07pt}\bzdrv{}},
```

the slanted lines are revived to give complete formulas of benzene, as shown in the right column of Table 1.

3.2 Size Reduction of heterocycles

Table 2 shows the effect of size reduction to the drawing of 4-chloropyridine, where `\unitlength` is changed from 0.1pt (default value) to 0.04pt by using `\changeunitlength`.

3.3 Nested substitution

Formulas with nested substitution can be completely reduced in size by the following code:

```
\changeunitlength{0.07pt}\scriptsize
\decaheterov[] {4a==N}{4D==O;7B==HO;%
{{10}A==H,%
```

Table 1: With and without `sizeredc.sty`

without <code>sizeredc.sty</code> (Version 2.00)	with <code>sizeredc.sty</code> (Version 3.00)
0.08pt	0.08pt
0.07pt ^a	0.07pt
0.06pt ^a	0.06pt

^aSlanted inner bonds disappear.

Table 2: Size reduction of 4-Chloropyridine

0.1pt ^a	0.08pt ^b	0.07pt ^c
0.06pt ^c	0.05pt ^d	0.04pt ^d

^aA standard size.

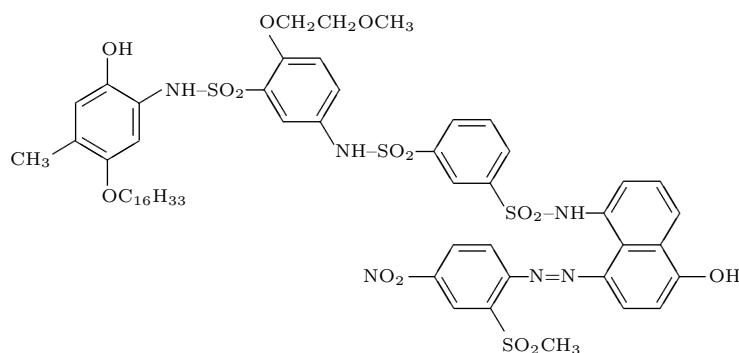
^bThe font size is set by `\small`

^cThe font size is set by `\scriptsize`

^dThe font size is set by `\tiny`

```
\changeunitlength{0.07pt}
\scriptsize
\bzdrv{1==OH;5==CH$_{3}$;4==OC$_{16}$H$_{33}$H$_{33}$;2==\ryl(4==NH--SO$_{2}$$_{2}$)$%
{4==\bzdrh{1==(y1);2==OCH$_{2}$$_{2}$CH$_{2}$OCH$_{3}$};%
5==\ryl(2==NH--SO$_{2}$$_{2}$)$\{4==\bzdrh{1==(y1);5==\ryl(2==SO$_{2}$$_{2}$)--NH)%%
{4==\naphdrh{1==(y1);5==OH;8==\lyl(4==N=N)\{4==\bzdrh{4==(y1);%
1==NO$_{2}$$_{2}$;5==SO$_{2}$$_{2}$CH$_{2}$$_{3}$}}}}}}}}}
```

(\changeunitlength{0.07pt})



(\changeunitlength{0.1pt})

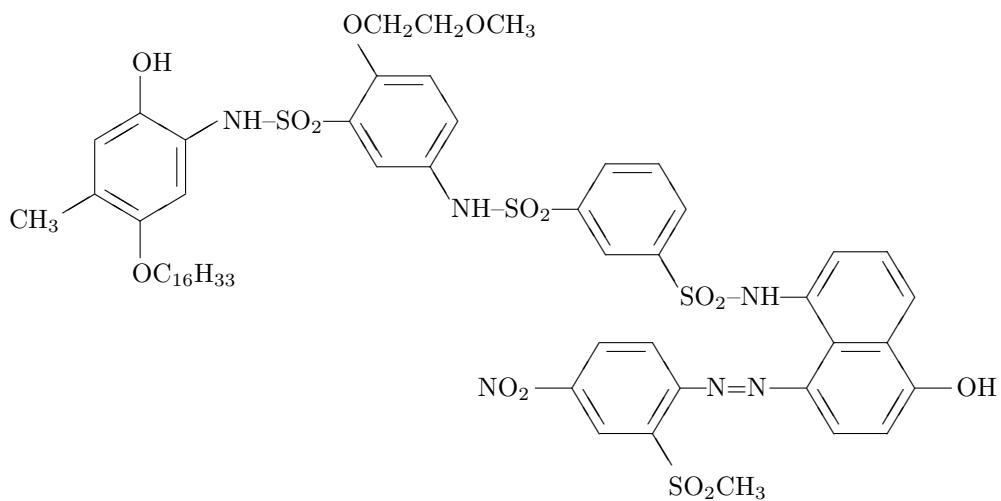
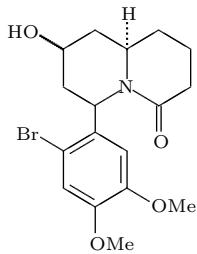


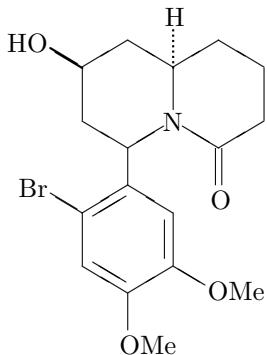
Figure 1: A cyan dye releaser drawn at unit lengths 0.07pt and 0.1pt

```
5==\bzdrv{3==OMe;4==OMe;6==Br;1==(y1)}
```

This code produces the formula shown below:



The formula of the standard dimension is drawn by the same code after returning to the default condition or by declaring `\changeunitlength{0.1pt}` explicitly.



A cyan dye releaser [9] has been drawn by using two or more `\ryl` and `\yl` commands, as shown in the on-line manual of X^MTEX Version 2.00 and has also been depicted in different ways (see Chapters 14 and 15 of the X^MTEXbook [10]). By virtue of the present version of X^MTEX, the size of the formula can be reduced with the code shown in Fig. 1. It should be emphasized that the portability of the XyMTeX system is still maintained in Version 3.00, where it is assured by the reliance on the L^AT_EX picture environment and the `sizeredc` package (a revision of `epic`).

References

- [1] Fujita S. & Tanaka N., "X^MTEX (Version 2.00) as Implementation of the X^M Notation and the X^M Markup Language", *TUGboat*, **21** (1), 7–14 (2000).
- [2] Fujita S. & Tanaka N., "X^M Notation for electronic communication of organic chemical structures", *J. Chem. Inf. Comput. Sci.*, **39**, 903–914 (1999).
- [3] Fujita S., "X^M Markup Language (X^MMML) for electronic communication of chemical documents containing structural formulas and reaction schemes", *J. Chem. Inf. Comput. Sci.*, **39**, 915–927 (1999).
- [4] Lamport L., *L^AT_EX. A document Preparation System*, 2nd ed. for L^AT_EX 2_E, Addison-Wesley, Reading (1994).
- [5] Goossens M., Mittelbach F., & Samarin A., *The L^AT_EX Companion*, Addison-Wesley, Reading (1994).
- [6] For epic macros, see Podar S., "Enhancements to the picture environment of L^AT_EX", Manual for Version 1.2 dated July 14, 1986.
- [7] For the portability of graphic applications of T_EX, L^AT_EX and relevant systems, see Goossens, M., Rahtz, S., & Mittelbach, F., *L^AT_EX Graphics Companion*, Addison Wesley Longman, Reading (1997).
- [8] The system is now available from Fujita's homepage via the Internet:
<http://imt.chem.kit.ac.jp/fujita/fujitas/fujita.html>
A detailed manual is also available from this homepage.
- [9] Fujita S., Koyama K., & Ono S., "Dye Releasers for Instant Color Photography", *Rev. Heteroatom Chem.*, **7**, 229–267 (1992).
- [10] Fujita S., *X^MTEX—Typesetting Chemical Structural Formulas*, Addison-Wesley, Tokyo (1997). The book title is abbreviated as "X^MTEXbook" in the present article.

◇ Shinsaku Fujita
Department of Chemistry and
Materials Technology,
Kyoto Institute of Technology,
Matsugasaki, Sakyo-Ku, Kyoto,
606-8585 Japan
fujitas@chem.kit.ac.jp

◇ Nobuya Tanaka
Department of Chemistry and
Materials Technology,
Kyoto Institute of Technology,
Matsugasaki, Sakyo-Ku, Kyoto,
606-8585 Japan
nobuya@chem.kit.ac.jp

The package `ps4pdf`: from PostScript to PDF

Rolf Niepraschk and Herbert Voß

Abstract

The only graphic object which `TeX` can handle internally is the `picture` environment, which is on the one hand very easy to use, but on the other hand very restrictive. All other graphical material must be encapsulated in `\special` commands and later extracted by the DVI processor, for example, `dviips` into PostScript code. Packages like `pstricks` (and its extensions `pst-xxxx`) and `psfrag` can create such `\special` commands. Unfortunately, `pdflatex` cannot work when one of these packages is part of the document file. The new package `ps4pdf` makes it possible to collect all PostScript-related parts and convert them to PDF in a single run.

1 Introduction

PDF output can be created in several different ways:

- traditional: `dvi→ps→pdf` using the commands `latex` to create the DVI file, `dviips` for the `ps` file and `ps2pdf` for the PDF file.
- using `dvipdfm` to skip the `ps` step—but have a look at the manual page of `dvipdfm` for some restrictions.
- using `pdfLATEX` to skip the `dvi` step and generate PDF directly—but this has the problem stated in the abstract.
- using `VTEX` as an alternative to `pdfLATEX`—but this is available without charge only for Linux and OS/2 [2].
- using the package `pstricks` [5, 6]—but in some PostScript environments, a bounding box is difficult to determine.
- using the package introduced here, `ps4pdf` [4].

This new package `ps4pdf` works very differently from `pstricks`. It uses the package `preview`, which is part of the `latex-preview` [1, 3] bundle, available at any CTAN server. `preview` extracts all ‘marked’ parts of a complete `LATEX` document to a DVI file, in which each such part is saved on a separate page. This makes it easy to convert this DVI file into PDF format and then include these parts in a last `pdfLATEX` run.

2 Package options

Table 1 shows the available package options. Specifying `inactive` causes all the `ps4pdf` macros disables everything except the trimming functionality, so that `latex` runs in the usual way. This makes

direct PostScript output possible, so that the other methods listed above can be used.

Table 1: `ps4pdf` package options

name	meaning
<code>active</code>	enables the <code>ps4pdf</code> macros (default)
<code>inactive</code>	disables the <code>ps4pdf</code> macros, making direct PostScript output possible; this is the default if <code>VTEX</code> is detected
<code>trim</code>	modify the internal bounding box (similar to the <code>trim</code> option from <code>\includegraphics</code>)
<code>draft</code>	suppresses the content of the <code>\PSforPDF</code> macros (do not influence the content of the graphics container)
<code>final</code>	shows the content of the <code>\PSforPDF</code> macros (default)

3 Usage

3.1 Usage in preamble

Assume that we have the small `LATEX` file shown in listing 1; the output is shown in figure 1. To use the `ps4pdf` package, we must pass **all** PostScript-related parts through the `\PSforPDF` macro, beginning with the preamble. And of course we must use the `ps4pdf` package itself.

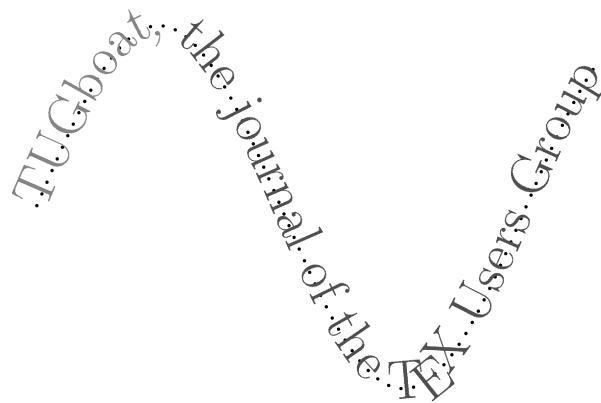


Figure 1: Output of listing 1

`ps4pdf` must know every package that the PostScript images depend upon, otherwise it cannot create the images and convert them to PDF. For our example here, listing 2 shows how this is done.

Listing 1: Demonstration of a `pstricks` object

Table 2: The command sequence from PostScript to PDF

> graphics container creation:	
<code>latex file.tex</code>	\LaTeX run
<code>dvips -Ppdf -o file-pics.ps file.dvi</code>	<code>dvips</code> run to convert the DVI file to PostScript
<code>ps2pdf file-pics.ps file-pics.pdf</code>	<code>ps2pdf</code> run to convert the PostScript file to PDF
> document creation:	
<code>pdflatex file.tex</code>	first <code>pdflatex</code> run
<code>bibtex file</code>	\BIBTeX run
<code>[...]</code>	any other additional runs (for example: glossary)
<code>pdflatex file.tex</code>	second (and last) <code>pdflatex</code> run

```

1 \documentclass{article}
2 \usepackage{pst-plot}
3 \usepackage{pst-text}
4 \begin{document}
5 \psset{unit=1cm}
6 \begin{pspicture}(-0.25,-2.25)(6.25,2.25)
7 \pstextpath[linestyle=none]%
8 {\psplot[linewidth=1pt,%
9 linestyle=dotted,%
10 plotpoints=300,%
11 xunit=0.015,%
12 yunit=2]{0}{400}{x sin}}
13 {\LARGE TUGboat, the journal%
14 of the \TeX{} Users Group}
15 \end{pspicture}%
16 \end{document}

```

Listing 2: Using `\PSforPDF` in the preamble

```

1 \documentclass{article}
2 \usepackage{ps4pdf}
3 \PSforPDF{\%--- BEGIN PSforPDF
4 \usepackage{pst-plot}%
5 \usepackage{pst-text}%
6 \%--- END PSforPDF

```

3.2 Usage in document body

For the user, there is no difference between using `ps4pdf` in the preamble or in the text part of the document: any PostScript-related material must be passed to the `\PSforPDF` macro. Only internally are these separate parts of the document handled in different ways. Thus, listing 1 is changed to what we have in listing 3.

Listing 3: Using `\PSforPDF` in the body

```

1 \begin{document}
2 \PSforPDF{\%--- BEGIN PSforPDF
3 \psset{unit=1cm}
4 \begin{pspicture}(-0.25,-2.25)(6.25,2.25)
5 [ ... ]
6 \end{pspicture}%
7 \%--- END PSforPDF
8 \end{document}

```

4 Implementation

`ps4pdf` is part of the process shown in table 2, which can also be encapsulated as a shell script (listing 4).

Listing 4: Shell script implementing table 2

```

1#!/bin/sh
2# build a pdf file with PostScript code
3# Herbert Voss 2003-03-10
4# usage: ps4pdf.sh file (without suffix tex)
5latex $1.tex
6dvips -Ppdf -o $1-pics.ps $1.dvi
7ps2pdf $1-pics.ps $1-pics.pdf
8pdflatex $1.tex
9bibtex $1
10pdflatex $1.tex

```

- In the first \LaTeX run, `preview-latex` extracts all objects which are included as a argument to `\PSforPDF`, and saves them into `<file>.dvi`. Each object is on its own page.
- This object file is then converted into a PostScript file with `dvips`. The `-Ppdf` option tells `dvips` to load the config file for PDF-related output. `dvips` creates the new file `<file>-pics.ps`.
- This PostScript file `<file>-pics.ps` is then converted into the corresponding PDF file with `ps2pdf` (a front end to Ghostscript).

- At this point the important work of ps4pdf is done, and the usual pdflatex runs can be done, as well as additional runs for BiBTeX or other post-processors.
- If all worked well, then the final PDF file includes all PostScript-related code as PDF images!

The meaning of the macro \PSforPDF changes for the pdfLATEX runs. It now becomes:

```
\includegraphics[page=<n>]%
{<file>-pics.pdf}
```

which inserts the *n*th page of the object file. An internal counter is used to get the right object at the right place. This implies that the whole command sequence in table 2 has to be repeated if the sequence of the objects changes.

5 Saving the images

Saving all graphical objects from the PDF graphic container as single files is very easy and can be done with the small script shown in listing 5. After running this script with <file>-pics.pdf as parameter, the images are saved as picture<n>.eps. This may be useful for other purposes.

Listing 5: Script to convert PDF images to PostScript

```
1 #!/bin/sh
2 File=$1
3 n='pdfinfo $File | awk '$1 ~ /Pages:/ {
4   print $2}''
4 for i in `seq $n` ; do
5   pdftops -f $i -l $i -eps $File picture$i.
      eps
6 done
```

References

- [1] David Kastrup. *preview-latex*. CTAN:/support/preview-latex/, 2003.
- [2] Micropress. *VTEX/Lnx*. http://www.micropress-inc.com/linux/, 2003.
- [3] Rolf Niepraschk. Anwendungen des LATEX-pakets preview. *Die TExnische Komödie*, 1/2003:60–65, February 2003.
- [4] Rolf Niepraschk. *ps4pdf*. CTAN:/macros/latex/contrib/ps4pdf/, 2003.
- [5] Chambert-Loir Radhakrishnan, Rajagopal. *pdftricks*. CTAN:/macros/latex/contrib/supported/pdftricks/pdftricks.sty, 2002.

- [6] Herbert Voß. *PSTricks Support for pdf*. http://www.pstricks.de/pdf/pdftricks.phtml, 2002.

◊ Rolf Niepraschk
Persiusstr. 12
10245 Berlin GERMANY
niepraschk@ptb.de

◊ Herbert Voß
Wasgenstr. 21
14129 Berlin GERMANY
voss@perce.de
http://www.perce.de

Instant Preview and the \TeX daemon

Jonathan Fine

Abstract

Instant Preview is a new package, for use with Emacs and `xdvi`, that allows the user to preview instantly the file being edited. At normal typing speed, and on a 225MHz machine, it refreshes the preview screen with every keystroke.

Instant Preview uses a new program, `dvichop`, that allows \TeX to process small files over 20 times quicker than usual. It avoids the overhead of starting \TeX . This combination of \TeX and `dvichop` is the \TeX daemon.

One instance of the \TeX daemon can serve many programs. It can make \TeX available as a callable function. It can be used as the formatting engine of a WYSIWYG editor.

This paper will demonstrate Instant Preview, describe its implementation, discuss its use with \LaTeX , sketch the architecture of a WYSIWYG \TeX , and call for volunteers to take the project forward.

Instant Preview at present is known to run only under GNU/Linux, and is released under the GPL. It is available at: <http://www.activetex.org>.

Instant Preview

\TeX is traditionally thought of as a batch program that converts text files into typeset pages. This article describes an add-on for \TeX , that in favourable circumstances can compile a file in a twentieth of

The screenshot shows two windows side-by-side. The left window is an Emacs buffer titled 'Screenshot.tex' with version 1.1 from 2001/05/27 at 16:45:52. It contains LaTeX code for 'Instant Preview'. The right window is an xdvil viewer showing the preview of the document. The code in the buffer includes comments explaining the setup: using \verb|\tex|, \verb|\verbblxdvil|, and \verb|\verbldvichop|, and how it avoids start-up costs by piping output to a pipe. It also mentions error recovery using \verb|\scrollmode|. The xdvil window shows a simple mathematical formula: $\sqrt{2} \approx 1.41$.

```

Buffers Files Tools Edit Search Mule Help
% $Id: screenshot.tex,v 1.1 2001/05/27 16:45:52 jfine Exp
\setcounter{section}{0} % reset with every keystroke
\section{Instant Preview Screen Shot}
Instant Preview is an add-on for \TeX. It works with almost all macro packages. Here we see it being used with \LaTeX. (Token mathematics:  $\sqrt{2} \approx 1.41$ )
\subsection{Components}
On the left, Emacs. On the right, \verbblxdvil. Behind the scenes, \TeX the program, some macros, and \verbldvichop (a new program). We're running GNU/Linux.
\subsection{Using it}
Type into the Emacs window. As you type, so the display in the \verbblxdvil window is refreshed, in real time.
\subsection{How it works (\verb|pageno|=\the\count0)}
Speed is of the essence. We start \TeX once, and its output goes to a pipe. This avoids start-up costs.
Each keystroke feeds \TeX, and \verbldvichop breaks the output up into small pieces, which it sends to \verbblxdvil.
\subsection{Error recovery}
To prevent \TeX from stalling on errors, \verb|\filbreak| we use \verb|\scrollmode|. % <-- spot the other error
% end of screenshot.tex

```

--** screenshot.tex (Fundamental Preview RCS:1.1) -->

Figure 1: Screen shot of Instant Preview.

the normal time. This allows \TeX to be used in interactive programs. This section describes Instant Preview. See figure 1 for a screen-shot.

Types of users Almost all users of \TeX are familiar with the edit-compile-preview cycle that is part of the customary way of using \TeX . Previewing is very useful. It helps avoid wasting paper, and it saves time. In the early days, it could take several seconds to compile and preview a file, and perhaps minutes to print it. Today it takes perhaps about a quarter of a second to compile and preview a file.

Many of today's newcomers to computing, and most users of WYSIWYG word processors, expect to have instant feedback, when they are editing a document. Users of \TeX expect the same instant feedback, when they are editing a source file in a text editor. Because they have absorbed the meaning of the markup codes, they can usually imagine without difficulty the printed form of the document. They know when the markup is right.

Beginners tend to compile the document frequently, because they are uncertain, and wish to have the positive reinforcement of success. Instant Preview, again under favourable circumstances, can reduce to a twentieth the time take to compile and preview a file. This makes it practical to offer pre-

1 Instant Preview Screen Shot

Instant Preview is an add-on for \TeX . It works with almost all macro packages. Here we see it being used with \LaTeX . (Token mathematics: $\sqrt{2} \approx 1.41$)

1.1 Components

On the left, Emacs. On the right, xdvi . Behind the scenes, \TeX the program, some macros, and dvichop (a new program). We're running GNU/Linux.

1.2 Using it

Type into the Emacs window. As you type, so the display in the xdvi window is refreshed, in real time.

1.3 How it works ($\text{pageno} = 155$)

Speed is of the essence. We start \TeX once, and its output goes to a pipe. *This avoids start-up costs.*

Each keystroke feeds \TeX , and dvichop breaks the output up into small pieces, which it sends to xdvi .

1.4 Error recovery

To prevent \TeX from stalling on errors, we use \scrollmode . % <-- spot the other error

view after every keystroke. Beginners will be able to see their failures and successes as they happen.

Experienced users do not need such a high level of feedback, and prefer to devote the whole screen to the document being edited. However, even experts have the same need for positive reinforcement, when they use a package that is new to them.

Modus operandi Here we describe three possible ways of using Instant Preview. At the time of writing, only the last has been implemented. We assume that the document is in the editing stage of its life cycle, or in other words the location of page breaks and the like is not of interest.

The expert needs only occasionally to preview the source document. She will select the region of interest, and ask for it to be previewed. Instant Preview here may provide a quick and convenient interface, but the operation is uncommon and so the functionality should be unobtrusive.

When doing something tricky, the user might wish to focus on a part of the document, and for this part have Instant Preview after every keystroke. The tuning of math spacing in a formula is an example. Few if any users invariably know, without looking, what tuning should be applied to a moderately complicated formula. This applies particularly to displayed equations wider than the measure,

multi-line equations, and commutative diagrams. It also applies to the picture environment (for which the special tool **TeXcad** was written).

For the beginner, everything is tricky, even straight text. The beginner hardly knows that `\{}`, `#`, `_`, `$` are all special characters, and that “‘ and ’’ are the way to get open and close double quotes. Even experts, who know full well the rules for spaces after control sequences, sometimes make a mistake¹. The absolute beginner is likely to want Instant Preview of everything, absolutely all the time. Later, with experience, the training wheels can be removed.

Implementation Instant Preview has been implemented using Emacs and `xpdf`. There seems to be no reason why another editor and previewer should not be used, provided the editor is sufficiently programmable, and the previewer can be told to refresh the file it is previewing.

Instant Preview works by writing the region to be previewed, together with suitable preamble and postamble, to a special place. From there, the `TeX` daemon picks it up, typesets it, and writes it out as a `dvi` file. Once written, the previewer is told to refresh its view of the `dvi` file.

The main difference between the three modes is what is written out, and when. Absolute beginner mode writes out the whole buffer, after every keystroke. Confident expert mode writes out a selected region, but only on demand.

At the time of writing (mid-June 2001), only absolute beginner mode has been implemented. Further progress requires above all clear goals and Emacs programming skills.

The dvichop program

For interactive programs, speed is of the essence. Therefore, we will look at TeX's performance. The author's computer has a 225MHz Cyrix CPU. So that we have a definite figure, we will say that on this machine a response time of 1/10 seconds is acceptable.

Typesetting `story.tex` There is a file, `story.tex`, that is part of every TeX distribution. It is described in *The TeXbook*. On the author's computer, the command

```
time tex ./story \\end
```

Mode	seconds
Console, output to /dev/null	.492
Console, output to screen	.507
X-windows, output to /dev/null	.497
X-windows, output to screen	.837

Table 1: Time taken to typeset `story.tex` 100 times

takes .245 seconds to execute². This seems to make Instant Preview impossible.

However, the command

```
time tex \\end
```

takes only .240 seconds to execute. Therefore, it takes \TeX almost 1/4 of a second to load and exit, while typesetting the two short paragraphs in *story.tex* can be done about 20 times in the target time of a tenth of a second.

Thus, provided the overhead of loading (and exiting) TeX can be avoided, Instant Preview is possible.

Remarks on performance The simple tests earlier in this article show that it takes TEX about 0.005 seconds to typeset the file `story.tex`. This subsection gives a more precise result. It also shows some of the factors that can influence apparent performance.

The file 100story.tex is as below.

```
\def\0{\input ./story }
\def\1{\0\0\0\0\0\0\0\0\0\0}
\def\2{\1\1\1\1\1\1\1\1\1\1}
\2 \end
```

Table 1 gives the time taken to process this file, in the various modes. It shows that on the author's machine and in the best conditions, it takes about $0.0025 \approx (0.492 - 0.240)/100$ seconds to process `story.tex` once.

Note that the time taken can be quite sensitive to the mode, particularly X-windows. We also note that using `\input story` (so that kpathsea looks for the file) adds about 0.025 seconds to the total time taken.

Starting \TeX once The solution is to start \TeX once, and use it to typeset multiple documents. Once \TeX has typeset a page, it uses the `\shipout` command to write it to the `dvi` file. The new page now exists on the file system, and can be used by other programs. Actually, this is not always true.

¹ In the first draft, the allegedly expert author forgot that & is also special, and also that \verb cannot be used in a LATEX footnote.

² To avoid the overhead of X-windows, this command was executed in a virtual console. The same goes for the other timing data. The input file is placed in the current directory to reduce kpathsea overheads.

To improve performance, the system dependent part of TEX usually buffers the output dvi stream. However, this can be turned off. We assume that dvi output is unbuffered.

Most dvi-reading applications are unable to process such an ill-formed dvi file. For example, most immediately seek to the end of the file, to obtain a list of fonts used. To bridge this gap, and thereby enable Instant Preview, the author wrote a utility program called dvichop.

This program takes as input a *dvi* file, perhaps of thousands of pages, and produces from it perhaps thousands of tiny *dvi* files. The little files are the ones that the previewer is asked to reload.

More exactly, dvichop looks for special *marker pages* in the output dvi-stream produced by TeX the program. The marker pages delimit the material that is to be written to the small dvi files. The marker pages also control where the output of dvichop is to be written, and which process is to be informed once the output page is ready.

Implementation The program `dvichop` is written in the *C* programming language. It occupies about 800 lines of code, and calls in a header file `dviop.h` to define the opcodes. A shell program `texd` starts \TeX and sends its `dvi` output to `dvichop`. More exactly, \TeX writes to a named pipe (a FIFO), which is then read by `dvichop`.

More on performance In the abstract it is claimed that **TEX** together with **dvichop** is over 20 times quicker than ordinary **TEX**, when applied to small files. Here is some test data to support this bold claim.

Normally, `dvichop` is run using a pipe. To simplify matters, we will create the input stream as a ordinary file. The plain input file listed below does this. It also illustrates the interface to `dvichop`.

```
% 100chop.tex
\newcount\dvicount
\def\0{
\begin{group} % begin chop marker page
  \global\advance\dvicount 1
  \count0\maxdimen \count1 3
  \count2 \dvicount \shipout\hbox{}
\endgroup
\input ./story % typeset the story
\begin{group} % end chop marker page
  \count0\maxdimen \count1 4
  \count2 0 \shipout\hbox{}
\endgroup
}
\def\1f{\0\0\0\0\0\0\0\0\0\0\0}
```

Typesetting `story.tex` 100 times in the conventional way takes approximately 24.5 seconds. Running `TEX` on `100chop.tex` takes about 0.510 seconds. This typesets the story for us 100 times. Running `dvichop` on the output file `100chop.dvi` takes 0.135 seconds. Its execution creates files `1.dvi` through to `100.dvi` that are for practical purposes identical to those obtained in the conventional way. The conventional route takes 24.5 seconds. The `dvichop` route took $0.510 + 0.135 = 0.645$ seconds.

This indicates that on `story.tex` using `dvichop` is $24.5/0.635 \approx 38$ times quicker. Some qualifying remarks are in order. In practice, using the pipeline will add overhead, but this seems to be less than 0.01 seconds. On the other hand, the present version of `dvichop` is not optimised.

The TEX daemon

At this point we assume the reader has some basic familiarity with client-server architecture. A server is a program that is running more or less continually, waiting for requests from clients. Clients can come and go, but servers are expected to persist. An operating system is a classic example of a server, while an application is a client.

Thanks for the memory Normally, **TEX** is run as an application or client program. It is loaded into memory to do its job, it does its job, and then it exits. In the mid-1980s, when the author started using a personal computer, having more than a megabyte of memory was uncommon. **TEX** is uncomfortable on less than 512Kb of memory. Thus running **TEX** as a server would consume perhaps half of the available memory. For all but the most rabid **TEX**-ophile, this is clearly not an option.

Today **TEX** requires perhaps 2Mb of memory, and personal computers typically have at least 32Mb of memory. Letting **TEX** remain in memory on a more or less permanent basis, much as Emacs and other programs remain loaded even when not used, is clearly practical. However, even today, for most

users there is probably not room to have more than a handful of instances of \TeX resident in memory.

Sockets The present implementation of Instant Preview uses a named pipe. Sockets provide a more reliable and flexible interface. In particular, sockets can handle contention (requests to the same server from several clients). Applications communicate to the X-server provided by X-windows by means of a socket.

Providing a socket interface to the \TeX daemon will greatly increase its usefulness. The author hopes that by the end of the year he or someone else will have done this.

\TeX as a callable function Over the years, many people have complained that the batch nature of \TeX makes it unsuitable for today's new computing world. They have wanted \TeX to be a callable function. However, to make \TeX a callable function, all that is required is a suitable wrapper, that communicates with the \TeX daemon.

At present the \TeX daemon is capable of returning only a *dvi* file. To do this, it must parse the output *dvi* stream. Suppose, for example, that the caller wants to convert the output *dvi* into a bitmap, say for inclusion in an HTML page. The present set-up would result in the *dvi* pages being parsed twice. Although this is not expensive, compared to starting up a whole new \TeX process, it is still far from optimal.

If the \TeX daemon could be made to load page-handling modules, then the calling function could then ask for the bitmap conversion module to handle the pages produced by the function call. This would be more efficient. However, as we shall soon see, premature optimisation can be a source of problems.

\TeX forever An errant application does not bring down the operating system. Strange keystrokes and mouse movements do not freeze X-windows. In the same way, applications should never be able to kill the \TeX daemon. To achieve this level of reliability is something of a programming problem.

One thing is clear: The application cannot be allowed to send arbitrary raw \TeX to the \TeX daemon. \TeX is much too sensitive. All it takes is something like

```
\global\let\def\undefined
```

and the \TeX daemon will be rendered useless.

A more subtle form of this problem is when a client's call to the daemon results in an unintended, unwelcome, and not readily reversible change of

state. For example, the L^AT_EX macro `\maketitle` executes

```
\global\let\maketitle\relax
```

which is an example of such a command. (Doing this frees tokens from \TeX 's main memory. When \TeX , macros and all, is shoe-horned into 512Kb, this may be a good idea.)

Protecting \TeX \TeX can be made a callable function by providing an interface to the \TeX daemon. Most applications will want an interface that is safe to use. In other words, input syntax errors are reported before they get to \TeX , and it is not possible to accidentally kill the \TeX daemon. To provide this, the interface must be well defined. For example, the input might be an XML-document (say as a string) together with style parameters, and the output would be say a *dvi* file. Alternatively, the input might be a pointer to an already parsed data structure.

In the long run, this interface is probably best implemented using compiled code, rather than \TeX macros. Once a function is used to translate source document into \TeX input, there is far less need for developers to write complicated macros whose main purpose is to provide users with a comfortable input syntax. Instead, the interface function can do this.

When carried out in a systematic manner, this will remove the problem that in general L^AT_EX is the only program that can understand a L^AT_EX input file. The same holds for other \TeX macros formats, of course. Note that Don Knuth's WEAVE (part of his literate programming system) is similarly compiled code that avoids the need to write complicated \TeX macros.

Visual \TeX

This article uses the term visual \TeX to mean programs and other resources that allow the user to interact with a document through a formatted representation, typically a previewed *dvi* file. We use it in preference to WYSIWYG (what you see is what you get) for two reasons. The first is today many documents are formatted only for screen, and never get printed. Help files and web pages are examples of this. The second is that even when editing a document for print, the user may prefer a representation that is not WYSIWYG.

In most cases the author will benefit from interacting with a suitably formatted view of the underlying document. The benefits of readability and use of space that typesetting provides in print also manifest on the screen. But to insist on WYSIWYG

is to ignore the differences between the two media. Hence our use of the term Visual \TeX .

Whatever term is used, the technical problems are much the same, which is how to enable user interaction with the dvi file.

Richer dvi files In Visual \TeX , the resulting dvi file is a view on the underlying document. For it to be possible to edit the document through the view, the view must allow the access to the underlying document. Editing changes applied to the view, such as insertion and deletion, can then be applied to the document.

Placing large numbers of \special commands in the dvi file is probably the best (and perhaps the only) way to make this work. Doing this is the responsibility of the macro package (here taken to include the input filter function described in the previous section). It is unlikely that any existing macro package, used in its intended manner, will support the generation of such enriched dvi files. The author's Active \TeX macro package[2] is designed to allow this.

Better dvi previewers Most dvi previewers convert the dvi into a graphics file, such as a bitmap. Some retain information about the font and position of each glyph. A text editor or word processor has a cursor (called point in Emacs), and by moving the cursor text can be marked. This is a basic property of such programs. So far as the author knows, no dvi previewer allows such marking of text.

Further reading This section is based on the author's article [1].

The Lyx editor for L^AT_EX adopts a visual approach to the generation of files that can be typeset using L^AT_EX. It does not support WYSIWYG interaction. Understanding the capabilities and limitations of Lyx is probably a good way to learn more about this area.

The next steps

This section discusses some of the opportunities and problems in this general area, likely to present themselves over the next year or two.

Applications Two areas are likely to be the focus of development in the next year or so. The first is the refinement of Instant Preview, as a tool for use with existing \TeX formats. Part of this is the creation of material for interactive (La) \TeX training. Instant Preview provides an attractive showcase for the abilities of \TeX and its various macro packages.

The second is \TeX as a callable function. This is required for Visual \TeX . One of the important missing components are libraries that allow rich interaction with dvi files. This will lay the foundation for \TeX being embedded in desktop applications.

Licence The work described this article is at present released under the General Public Licence of the Free Software Foundation (the GPL). Roughly speaking, this means that any derived work that contains say the author's implementation of the \TeX daemon must also be released under the GPL.

However, the \TeX daemon is the basis for \TeX as a callable function, and for good reason library functions are usually released under the Lesser (or Library) General Public Licence (the LGPL), or something similar. This means that the library as is can be linked into proprietary programs, but that any enhancement to the library must be released under the LGPL.

Porting \TeX runs on almost all computers, and where it runs, it gives essentially identical results. The same applies, of course, to \TeX macros. By and large, it is desirable that the tools used with \TeX run can be made to run identically on all platforms. This is not to say that the special features of any particular platform should be ignored. Nor is it to say that advances (such as Instant Preview itself) should not first manifest on a more suitable platform.

Cross-platform portability is one of the great strengths of \TeX . What is desirable is that programs that run with \TeX have a similar portability. Many people cannot freely choose their computing platform. If \TeX and friends are available everywhere, this make \TeX a more attractive choice.

In the 1980s, in the early days of \TeX , many pioneers ported \TeX to diverse platforms. This work established deep roots that even today continue to nourish the community. Although Instant Preview, even when fully developed, is not on the same scale as \TeX , it being ported will similarly nourish the community.

\TeX macros Visual \TeX requires a stable \TeX daemon, which in turn will require a macro package (or a pre-loaded format). This new use of \TeX places new demands on the macros. Here, we include in macros any input filter functions used to protect the \TeX daemon from errant applications.

These new demands include protection against change of state, reporting and recovery from errors, ability to typeset document fragments, support for rich dvi file, and the ability for a single daemon

to support round-robin processing of multiple documents. Once tools are in place, much of the input is likely to be XML, and much of the output will be for screen rather than paper.

The existing macros packages (such as plain, L^AT_EX and Con^TEXT) were not written with these new requirements in mind. Although they are useful now, in the longer term it may be better to write a new macro package from scratch, for use in conjunction with suitable input filters.

Summary

By running T_EX within a client-server architecture, many of the problems traditionally associated with it are removed. At the same time, new demands are placed on macro packages, device drivers (such as dvichop and xdvi) and a new category of software, input filters (such as WEAVE).

This new architecture allows Instant Preview, and opens the door to Visual T_EX. All this is possible without making any changes to T_EX the program, other than in the system dependent part.

Don Knuth In 1990, when he told us [4] that his work on developing T_EX had come to an end, Don Knuth went on to say:

Of course I do not claim to have found the best solution to every problem. I simply claim that it is a great advantage to have a fixed point as a building block. Improved macro packages can be added on the input side; improved device drivers can be added on the output side.

The work described in this article has taken its direction from this statement. One of the most obvious characteristics of today's computer monitors (not to be confused with the chalk monitor in classrooms of old) is their widespread use of colour. T_EX is clumsy with colour. T_EX was not designed with Visual T_EX in mind. However, we still have our hands full making the best of what we have with T_EX. If our labours bear fruit, then in time a place and a need for a successor will arise.

Again, this possibility was foretold by Don Knuth [3]:

Of course I don't mean to imply that all problems of computational typography have been solved. Far from it! There are still countless important issues to be studied, relating especially to the many classes of documents that go far beyond what I ever intended T_EX to handle.

References

- [1] Jonathan Fine, Editing .dvi files, or Visual T_EX, *TUGboat*, **17** (3) (1996), 255–259.
- [2] ———, Active T_EX and the DOT input syntax, *TUGboat*, **20** (3) (1999), 248–261
- [3] Donald E. Knuth, The Errors of T_EX, *Software—Practice & Experience*, **19** (1989) 607–685 (reprinted in *Literate Programming*)
- [4] ———, The future of T_EX and METAFONT, *TUGboat*, **11** (4) (1990), 489 (reprinted in *Digital Typography*)

◊ Jonathan Fine
203 Coldhams Lane, Cambridge,
CB1 3HY, UK
jfine@activetex.org

Graphics

Space geometry with METAPOST*

Denis Roegel

Abstract

METAPOST is a tool especially well-suited for the inclusion of technical drawings in a document. In this article, we show how METAPOST can be used to represent objects in space and especially how it can be used for drawing geometric constructions involving lines, planes, as well as their intersections, orthogonal planes, etc. All the features belong to a new METAPOST package aimed at all those who teach and study geometry.

This article is dedicated to Donald Knuth whose PhD dissertation was on projective geometry.

1 Introduction

METAPOST (Hobby, 1992; Goossens, Rahtz, and Mittelbach, 1997; Hoenig, 1998; Hagen, 2002) is a graphical description language created by John Hobby from the METAFONT system (Knuth, 1986).

* Translated from “La géométrie dans l'espace avec METAPOST,” *Cahiers GUTenberg* 39–40, May 2001, pages 107–138, with permission.

A two-dimensional drawing is represented as a program which is compiled into a PostScript file. A drawing can be described in a very precise and compact fashion by taking advantage of the declarative nature of the language. For instance, linear constraints between the coordinates of several points, such as those of a central symmetry, are expressed very naturally by equations. Furthermore, it is possible to manipulate equations involving values that are not completely known. For instance, in order to express that p_3 is the middle of $[p_1, p_2]$, it suffices to write: $p_3-p_1=p_2-p_3$, or $p_3=.5[p_1, p_2]$.

When this equation is given, some and possibly all of the coordinates of the three points may be unknown. Taking the equation into account represents the addition of a constraint. Constraints are added until the values involved are precisely known. In the previous example, the three points can be completely determined by positioning p_1 and p_2 . A value can remain indetermined as long as it is not involved in a drawing. Finally, METAPOST alerts the user if there are redundant or inconsistent equations.

2 A first example in plane geometry

In order to get a good understanding of how METAPOST can naturally express a geometric problem, let us study the representation of a triangle property, such as the existence of the *nine points circle* (first stated by Poncelet and Brianchon in 1821). Figure 1 shows the result produced by METAPOST. This example will also serve as an introduction to METAPOST for the reader discovering the language here.

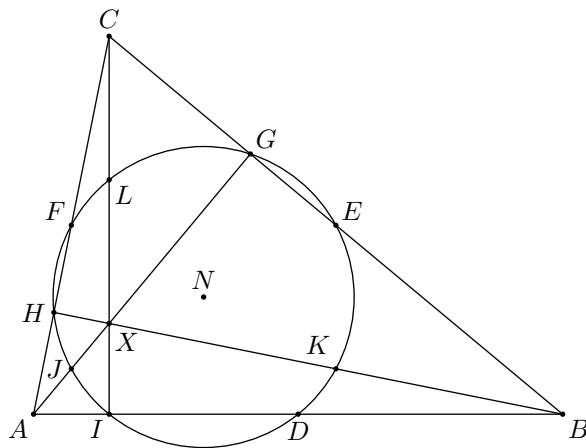


Figure 1: The nine points circle.

In this figure, we have first defined the points, then we set the three vertices of the triangle as functions of the origin (`origin`) and an arbitrary unit u

enabling us to easily change the size of the graphics later on:¹

```
numeric u; u=1cm;
pair A,B,C,D,E,F,G,H,I,J,K,L,N,X;
A=origin; B-A=(7u,0); C-A=(u,5u);
```

Then, the middles D , E and F of the triangle's sides are determined by equations:

```
D=.5[A,B]; E=.5[B,C]; F=.5[A,C];
```

G , H and I are the feet of the triangle's heights. They can be obtained easily by computing the intersection of a side with a segment starting at the opposite vertex and directed toward a direction at right angle with the opposite side. The `whatever` definition is especially useful in this case, since it represents an anonymous unknown (so that several occurrences of `whatever` do not represent the same unknown!). We have thus:

```
G=whatever[B,C]
=whatever[A,A+((C-B) rotated 90)];
```

This means that G is somewhere on (BC) and also somewhere on (AP) , where P is a point on the height. METAPOST gives a value to *both* unknowns in order to fulfill this equation. Similarly,

```
H=whatever[A,C]
=whatever[B,B+((C-A) rotated 90)];
I=whatever[A,B]
=whatever[C,C+((B-A) rotated 90)];
```

The orthocenter (intersection of the heights) is obtained with `intersectionpoint`. The $A--G$ construction represents the $[AG]$ segment:

```
X=(A--G) intersectionpoint (C--I);
```

The middles J , K and L of $[AX]$, $[BX]$ and $[CX]$ are obtained as were D , E and F previously:

```
J=.5[A,X]; K=.5[B,X]; L=.5[C,X];
```

Finally, in order to find the center N of the nine points circle (assuming its existence), it is sufficient to compute the intersection of two perpendicular bisectors, for instance those of $[ID]$ and $[DH]$:

```
N=whatever[.5[D,I],
(.5[D,I]+((D-I) rotated 90))]
=whatever[.5[D,H],
(.5[D,H]+((D-H) rotated 90))];
```

The circle's radius is found with:

```
r=arclength(I--N);
```

The triangle as well as the heights and the circle (centered on N and of diameter $2r$) are drawn with:

```
draw A--B--C--cycle;
draw A--G; draw B--H; draw C--I;
draw fullcircle scaled 2r shifted N;
```

The points are marked with `drawdot` after the line width has been increased. Finally, the annotations are all obtained on the model of:

¹ For the points, we could also have used `z0`, `z1`, etc., which are predefined variables.

```
label.top(btex $C$ etex,C);
```

The `label` instruction allows for the inclusion of `TEX` labels.

This example reveals the natural expression of geometric constraints for problems in plane geometry. All the constructions we have used are absolutely standard in `METAPOST`. Of course, if we had many such figures, we would introduce functions for the computation of the heights, the perpendicular bisectors, etc.

3 METAPOST extensions

`METAPOST` is an extensible system. At the basis, it is a program which loads an initial set of macros. It is then possible to add new domain-specific definitions. For instance, when we worked on the plane representation of objects in space, we developed a `3d` package, initially in order to manipulate polyhedra (Roegel, 1997). We have recently developed other extensions resting on the `3d` package. We view these extensions as “modules” of the `3d` package. In particular, we wanted to manipulate objects other than polyhedra, such as curves defined by equations, or given by a sequence of points. Among the extensions created, we have created a module providing various functionalities adapted to space geometry. This module, introduced here, is the `3dgeom` module² (see figure 2).

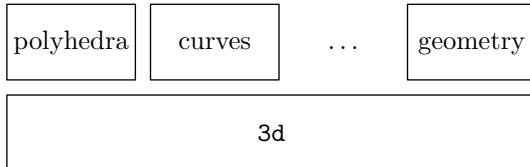


Figure 2: Structure of the `3d package` and of modules.

Some of the modules automatically load other modules. The `3dgeom` module loads for instance `3d`. A module is loaded only once.

A program using `3dgeom` will therefore start with input `3dgeom`.

4 Space geometry

4.1 A simple example

We will start by representing an elementary object, the cube (figure 3). For that, we will give the coordinates of its eight vertices.

The `3d` package defines a concept of point or vector as a triple of numerical values. The points

² This module is available on CTAN under `graphics/metapost/macros/3d`.

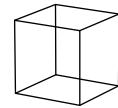


Figure 3: A cube shown in linear perspective.

must be defined by an allocation mechanism and must be freed when they are no longer used. The allocation of a point (resp. a vector) is done with `new_point` (resp. `new_vec`). This is a macro taking a point (resp. vector) name and allocating a memory area to store it. Freeing a point or a vector is done with `free_point` or `free_vec`, by giving the point or vector identification as a parameter. Hence, a program that wishes to use a vector `v` will look like this:

```
new_vec(v);
...
free_vec(v);
```

The set of all points and vectors is stored internally in a stack. Allocating or freeing a vector merely changes the stack pointer. As a consequence, points and vectors must be freed in the reverse order of their allocation. If that order is not respected, an unallocation error is raised.

```
new_point(pa); new_point(pb);
...
free_point(pb); free_point(pa);
```

It is not compulsory to free vectors or points, but not doing so will often have dramatic consequences if the allocations are within loops.

In order to ease the manipulation of sets of points, arrays can be allocated with `new_points` and freed with `free_points`. An array defined in that way has a name and a number of elements `n`. The elements are numbered from 1 to `n`. In our example, in order to create a cube, we declare a `vertex` array of eight points:

```
new_points(vertex)(8);
...
free_points(vertex)(8);
```

Each vertex is declared with the `set_point_` command, for instance:

```
set_point_(vertex1)(0,0,0);
```

By default, the perspective is linear (or central) and we have a camera witnessing the scene (figure 4). The camera must also be set. It corresponds to the predefined point `Obs`. Its position can be defined in space with `set_point_`. Moving the camera is done by expressing its coordinates in a parametric way, for instance:³

³ In the code, `cosd` and `sind` represent the trigonometric functions with arguments in degrees.

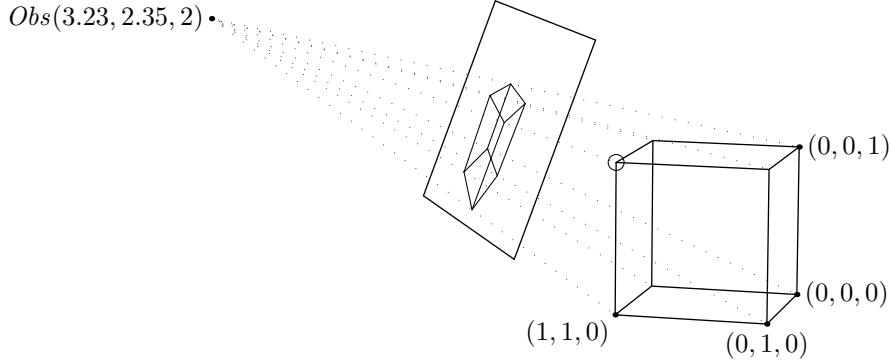


Figure 4: A cube and its projection on the screen. The focused point is circled.

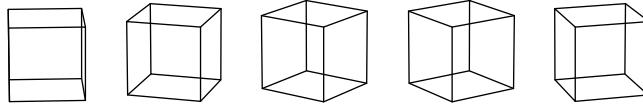


Figure 5: Five views of the cube “animation.”

```
set_point_(Obs)(20*cosd(3.6*i),20*sind(3.6*i),6);
```

By varying i , the camera goes through points of the circle $\mathcal{C}(t) = (20 \cos(3.6t), 20 \sin(3.6t), 6)$. Figure 5 shows five views of that sequence, for $i = 0, 5, 10, 15$ and 20 . An animation can be obtained by creating a sufficient amount of close views and by transforming the METAPOST outputs into GIF files. The procedure is explained in detail in our first article (Roegel, 1997).

Besides the camera position, it is necessary to define its orientation. It can be specified by three angles, but also in a simpler way by indicating a point towards which the camera is oriented and an angle. In order for the camera to be constantly focused on vertex 8, with the angle 90 (this angle corresponds to the degree of freedom of a rotation around the direction of view), it suffices to write:

```
Obs_phi:=90;
point_of_view_abs(vertex8,Obs_phi);
```

The “`:=`” assignment is used because it makes it possible to change the value of a variable when this variable already has a value. Writing `Obs_phi=90` can produce an error if `Obs_phi` is already set, and in particular if it has a value different from 90.

Finally, in order to define the view completely, the position of the screen must be given. In linear perspective, the screen is a plane orthogonal to the viewing direction. It is on the screen that the points in space are projected. Figure 4 shows that the focused point lies in the middle of the screen. The

screen is determined by its distance to the camera. This distance should also be that from which the computed scene is looked at (provided the scene is not scaled). We take for instance:

```
Obs_dist:=2;
```

At the time of projection, this value as well as the other coordinate values are multiplied by the value of `drawing_scale`, which defaults to 2 cm. The above value of `Obs_dist` therefore corresponds to a camera located at a distance of 4 cm from the screen.

Once the cube’s vertices and the camera are in place, the points must be projected on the screen with the `project_point` command. To each point in space corresponds a point in the plane. With `project_point(3,vertex3)`, point z_3 of the plane is associated to vertex 3 of the cube. Once the points have been projected, the edges can be drawn:

```
draw z1--z2--z4--z3--cycle;
draw z5--z6--z8--z7--cycle;
draw z1--z5; draw z2--z6;
draw z3--z7; draw z4--z8;
```

The complete program, with a few more initializations as well as the loop creating the animation, is given in figure 6. (This example didn’t make use of the geometry module.)

4.2 Improvements to the previous example

The source code producing the cube is rather simple, but it is easy to come up with drawings that

```

input 3danim; drawing_scale:=10cm;
new_points(vertex)(8);
for i:=0 upto 20:
beginfig(100+i);
% Cube
set_point_(vertex1)(0,0,0); set_point_(vertex2)(0,0,1);
set_point_(vertex3)(0,1,0); set_point_(vertex4)(0,1,1);
set_point_(vertex5)(1,0,0); set_point_(vertex6)(1,0,1);
set_point_(vertex7)(1,1,0); set_point_(vertex8)(1,1,1);
% Observer/camera
set_point_(Obs)(20*cosd(3.6*i),20*sind(3.6*i),6);
Obs_phi:=90; Obs_dist:=2; point_of_view_abs(vertex8,Obs_phi);
% Projections
for j:=1 upto 8:
project_point(j,vertex[j]);
endfor;
% Lines
draw z1--z2--z4--z3--cycle; draw z5--z6--z8--z7--cycle;
draw z1--z5; draw z2--z6; draw z3--z7; draw z4--z8;
endfig;
endfor;
free_points(vertex)(8);
end.

```

Figure 6: Program producing the cube animation.

are more complex and very difficult to handle, be it only because of the large amount of points involved. Moreover, the points do not necessarily belong to the same objects (we can have a cube, a tetrahedron, or other objects all present at the same time) and they may be subject to different treatments. It is in this spirit that we have introduced in the 3d package notions of *classes* and *objects* making it possible to group a number of points, in order to manipulate them globally, or in order to instantiate certain classes several times (see (Meyer, 1997) for more details on object-oriented programming). We will therefore redefine the cube, as a class, and then instantiate it.

The new code (figure 7) shows the definition of a “C” class. All the objects of that class are cubes. The class definition is split in three parts, which have to be called `def_C`, `set_C_points` and `draw_C`:

- The general definition function is `def_C`: This function takes as a parameter an object name and instantiates it. Specifically, this function defines the number n of points making up the object (they will be numbered 1 to n) and calls the function defining the points. Depending on the nature of the defined objects, it may perform other initializations; in particular, though it is not the case here, the initialization can de-

pend on the object name, that is, on the parameter.

- `set_C_points`, the function defining the points, takes the calls to `set_point_` but replaces them with `set_point`. Calling `set_point(1)(0,0,0)` means that point 1 of that object is defined. Thus, we have now a *local* point notion.
- Finally, a drawing function `draw_C` indicating how the object must be drawn.

The instantiation itself, that is the operation associating an object to a class, is done with a call to `assign_obj("cube", "C")`. The latter operation defines the *cube* object as an instance of the *C* class. It leads in particular to the call of the `def_C` function, hence to the computation of the cube’s points. It should be noted that in this example the points of the cube are set *before* the camera is set. In more complex drawings (like in figure 21), it is sometimes necessary to have points of an object depending on the position of the camera. In that case, besides the call to the `assign_obj` function (which must only occur once per object), the object positions can be recomputed with `reset_obj` (`set_C_points` cannot be used directly since this function doesn’t state how the absolute point numbers should be computed).

```

input 3danim; drawing_scale:=10cm;

vardef def_C(expr inst)=
  new_obj_points(inst,8); set_C_points(inst);
enddef;

vardef set_C_points(expr inst)=
  set_point(1)(0,0,0); set_point(2)(0,0,1); set_point(3)(0,1,0); set_point(4)(0,1,1);
  set_point(5)(1,0,0); set_point(6)(1,0,1); set_point(7)(1,1,0); set_point(8)(1,1,1);
enddef;

vardef draw_C(expr inst)=
  draw_lines(1,2,4,3,1); draw_lines(5,6,8,7,5); draw_line(1,5);
  draw_line(2,6); draw_line(3,7); draw_line(4,8);
enddef;

assign_obj("cube","C");

for i:=0 upto 20:
  beginfig(100+i);
    % Camera
    set_point_(0bs)(20*cosd(3.6*i),20*sind(3.6*i),6);
    Obs_phi:=90; Obs_dist:=2; point_of_view_obj("cube",8,Obs_phi);
    draw_obj("cube");
  endfig;
endfor;

end.

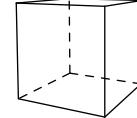
```

Figure 7: “3d object” code of the cube.

The main loop is now almost empty. The definitions concerning the camera have not been modified, except that concerning the focused point. The `point_of_view_obj` function is now used to aim an object point, here the cube point 8, and not an absolute point.

Finally, the cube is drawn with `draw_obj`. This command does both the projection and calls the `draw_C` command (with the "cube" parameter) and it is therefore not sufficient to call `draw_C("cube")`. The projection imposes a correlation between an object's local points and those of the plane. It is no longer possible to automatically associate point 7 of an object to z_2 , for instance.

In the sequel, we will always encapsulate our constructions in classes, even if (like here) we instantiate the class only once. The table below summarizes the main functions acting on objects.

**Figure 8:** The cube with a camera five times closer as in figure 3.

Definition of C class	<code>def_C et set_C_points</code>
Drawing definition of C class	<code>draw_C</code>
Object instantiation	<code>assign_obj</code>
Operations	<code>translate_obj</code> <code>rotate_obj</code> <code>scale_obj</code> <code>reset_obj</code>
Object drawing	<code>draw_obj</code>

4.3 Perspective

By default, all representations are done in linear (or central) perspective, that is the perspective corresponding to what a camera sees when its field of view is projected on a plane orthogonal to the viewing direction (Le Goff, 2000). The legitimate construction

rules of a linear perspective drawing have been codified by the Quattrocento painters and architects. This perspective is not very apparent on figure 3, because the camera is far from the cube (we are more than 2 meters away from a cube with a 10 cm side). If the camera gets close (and if `drawing_scale` is decreased to prevent the drawing from becoming too large) we obtain (with no changes to the cube) figure 8. The linear perspective shows clearly the vanishing lines.

In this example, the hidden edges have been dashed. The cube being defined as in the figure 7, it is not possible to determine automatically what is visible and what is not, since nothing has been said about the faces. But if the cube is defined as a polyhedron with the `3dpoly` extension (Roegel, 1997), the removal of hidden faces can be done automatically. However, this removal is for the moment only implemented for isolated convex objects. In the present article, all dashed lines are inserted manually.

Other perspectives are provided when the value of `projection_type` is changed. 0 corresponds to the linear perspective. 1 corresponds to a parallel perspective, where all projections are done parallel to the viewing direction and orthogonally to the projection plane. This perspective is different from the cavalier drawing. It corresponds to a camera set at an infinite distance, but looking at the scene with a telescope of infinite power. Since this perspective doesn't change the sizes, it is usually necessary to reduce the size of the projection by decreasing `drawing_scale`.

A first category of parallel projections are the isometric (also called military) perspective, dimetric and trimetric projections, all usually grouped under the axonometric perspectives. However, according to Krikke (Krikke, 2000), these projections are misnamed. Whereas the isometric perspective was invented by William Farish in 1822 to fulfill needs created by the industrial revolution, the real axonometric perspective is a perspective that originated in China and Japan, in particular because it is well suited to a presentation in rolls. In all parallel perspectives, the appearance of an object depends only on its orientation, not on its distance. A distant object doesn't appear smaller than a close object.

A value of 2 for `projection_type` corresponds to the oblique perspectives (which are parallel perspectives), but where the projection plane is not necessarily orthogonal to the projection direction. In general, the projection plane is chosen parallel to one of the object's faces. The most common oblique perspectives are the cavalier drawing and the cabinet

drawing. The *asian axonometry*, where an horizontal axis is orthogonal to the viewing direction, also seems to be an oblique perspective.

The `3d` package makes it possible to obtain any of these perspectives. In the case of parallel perspectives, the camera position is only used to find the viewing direction, not how close the view is. In the case of oblique projections, the projection plane being distinct from the camera, it is necessary to give it explicitly.

Figure 9 summarizes the various parallel perspectives.

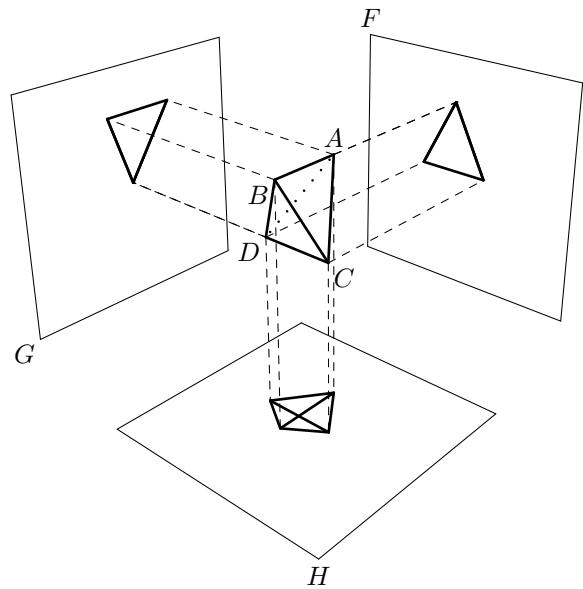


Figure 10: Orthogonal projections.

Another type of representation uses orthogonal (or orthographic) projections that are complementary. An object is described by several orthogonal projections on planes which are themselves orthogonal (see figure 10). The use of several orthogonal views and their crosschecking is known since antiquity, but have been expounded geometrically for the first time by Piero della Francesca in the Renaissance (cf. (Le Goff, 2000, p. 70)). The crosschecking of orthogonal views can be used to build a linear perspective view. For instance, in figure 10, if the two projections on planes F and G are represented in the same plane, the central projection on the plane H can be determined by crosscheckings. Albrecht Dürer used this technique to build the intersection of a cone and a plane in a famous engraving (Dürer, 1525).

Later on, Gaspard Monge systematically studied the method of double projection and codified it

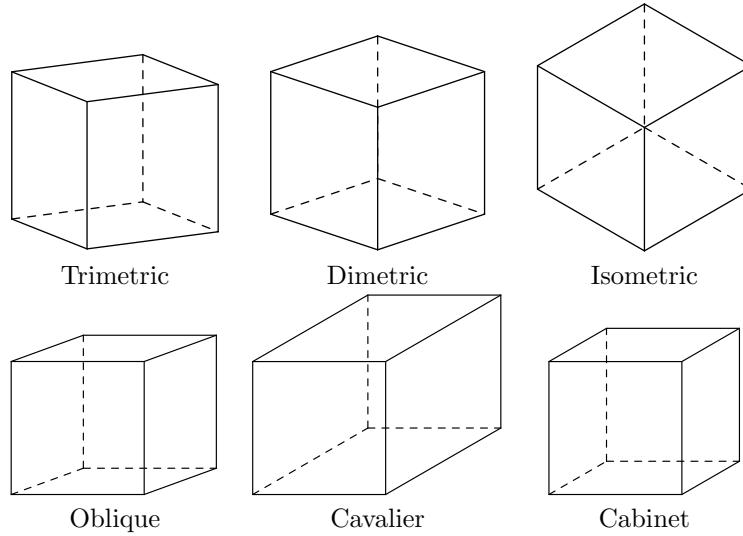


Figure 9: Parallel or perspective projections. The projections in the first row are projections where the projection plane is orthogonal to the projection direction. These projections are also named axonometric by certain authors (Gourret, 1994). The projections in the second row are oblique projections, where the projection plane is not orthogonal to the projection direction. The trimetric projection is the usual case of axonometric projection. In the dimetric projection, two of the axes are at the same scale and in the isometric projection, the three axes are represented at the same scale. In the oblique projections, one of the faces of the object is in general parallel to the projection plane and appears therefore without deformation. In the three cases shown, one of the faces of the projected cube is indeed a square. The angle of the vanishing lines varies. In the cavalier projection, the vanishing lines are represented at the same scale as the lines in the projection plane, which confers a non-natural aspect to that projection (even though it is a genuine oblique projection with no deformation). In order to get a natural feeling, one should observe the drawing from the right. In the cabinet projection, the vanishing lines are represented at 1/2 scale which gives a more natural representation, even though there is no vanishing point. Certain oblique projections are also called planometric, for instance when the face parallel to the projection plane represents a floor plan and when the vertical lines appear vertical in projection.

in his *Géométrie descriptive* (1799). His construction method is sometimes called Monge construction or Monge projection. The developments of this technique led to the standard representation in technical drawings in France, where a front face of an object is displayed, to its left the view on the right, to its right the view on the left, below the view seen from the top, etc. (In the U.S., the order is reversed, and the left-side view is on the left, the right-side view is on the right, etc.) The `3d` package currently does not provide an automatic means of representing these perspectives, but they can be simulated easily.

4.4 Local and absolute point indices

We have seen that each point or vector defined with `new_point` or `new_vec` corresponds to a stack element. A point is then given by its index in that

stack. The local index of a point is that point's number as it appears in the object definition, if it has been defined as an object point. When we defined the point 2 of our cube with `set_point`, 2 was not this point's index in the stack, but an index relative to the beginning of that object in the stack.

In certain cases, it is necessary to know the absolute index of a point, for instance because certain functions need it. This absolute index is obtained by the `pnt` function. For instance, in order to compute the middle of two points 1 and 2 and put the value in point 3, all these points being local points of an object, one way is to write:

```
vec_sum_(pnt(3),pnt(1),pnt(2));
vec_mult_(pnt(3),pnt(3),.5);
```

This is because `vec_sum_` (sum of two vectors) and `vec_mult_` (scalar multiplication of a vector)

**Figure 11:** Two vector operations.

(see figure 11) take as parameters absolute indices. It is not sufficient to create variants of these functions for the cases where these points (vectors) are local (these variants do exist and are called here `vec_sum` and `vec_mult`) because there are often intermediate cases involving local points from one or several objects, as well as points given by their absolute index (like for instance `Obs`). As a consequence, we have provided variants for the most common functions, but not for all of them. In certain cases, one has to resort to the `pnt` function. However, for the previous example, the `mid_point` function can be used:

```
mid_point(3,1,2);
```

The non-local (absolute) variant of `mid_point` is `mid_point_`. The more “internal” functions from the `3d` package have this final “`_`”. Thus, making the call `mid_point(3,1,2)` is equivalent to calling `mid_point_(pnt(3),pnt(1),pnt(2))`.

If we had wanted to define a non-local point p (defined outside an object) as being the middle of two local points, we would have written:

```
mid_point_(p,pnt(1),pnt(2));
```

In all cases, we should be careful to use `pnt` only inside an object (this makes it possible not to mention the object explicitly in the above examples) and more precisely only in the functions `def` and `draw` of that object.

4.5 Space structures

The objects definable with the `3d` package are in general rather complex objects which will end up being projected and drawn. These objects are not specially suited for a mathematical treatment. However, geometrical constructions, be it in the plane or in space, involve simple concepts such as lines, planes and other mathematically defined surfaces or volumes. These concepts, which we call here structures, are often used as intermediates for finding new points or new curves. The structures are seldom drawn. We will never draw a line, but only a seg-

ment of a line. We will never draw a plane, but only for instance a rectangle in that plane, or merely a few points in that plane.

In order to facilitate the manipulation of these structures, we have created them in a different and simpler fashion than the objects. These structures bear some similarities to the primitive types of Java. Each structure could be wrapped in an object or associated with an object, but we don’t do it here.

The structures are defined in the `3dgeom` module. They must also be allocated and freed.

4.5.1 Lines

The simplest of the structures we define is the line. A line is defined with two points. For instance, in order to define the line l going through local points 4 and 6, we write:

```
new_line(1)(4,6);
```

This function (abs. vers. `new_line_`) memorizes the two points, so that a later modification of these points does not modify the line. The line is therefore only initially attached to the points. However, this is seldom a problem for the structures are often introduced locally in a construction. Moreover, it is possible to create a version of `new_line` that does not duplicate the points.

Sometimes we want to define a line whose points have not yet been computed. Sometimes also, we would like to define a line using another pair of points, in order to start a different construction. The `set_line` (or `set_line_`) command can then be used:

```
set_line(1)(4,8);
```

Finally, when a line is no longer needed, it can be freed with `free_line` (which has only one version):

```
free_line(1);
```

It should be observed that the structures defined in an object must always be freed within that object and, as with the points, in the reverse order of their allocation.

4.5.2 Planes

A plane is defined in a manner analogous to a line, but using three points:

```
new_plane(p)(i,j,k);
set_plane(p)(i,j,k);
free_plane(p);
```

`new_plane_` and `set_plane_` are the absolute versions of these functions.

4.5.3 Other structures

Other structures (in the plane or not, such as circles, spheres, etc.) are defined in `3dgeom`, but they have not yet all been developed. It is very easy to add new structures and functions manipulating them.

4.6 Elementary constructions

4.6.1 Plane definitions

The use of structures tremendously simplifies geometric constructions in space. For instance, in order to draw the projections of the tetrahedron vertices in figure 10, we have defined the three projection planes with `new_plane`:

```
new_plane(f)(9,10,11);
new_plane(g)(13,14,15);
new_plane(h)(5,6,7);
```

4.6.2 Perpendiculars of a plane

We have then obtained the perpendiculars of these planes going through the object points, using the `def_vert_pl` function:

```
def_vert_pl(17)(1)(h);
def_vert_pl(18)(2)(h);
def_vert_pl(19)(3)(h);
def_vert_pl(20)(4)(h);
...
```

This function takes a point and a plane and determines the foot of the perpendicular to the plane going through the point given as parameter (here the second parameter).

4.6.3 Intersections between lines and planes

One of `3dgeom`'s functions computes the intersection of a line and a plane:

```
boolean b;
b:=def_inter_p_l_pl(i)(1)(p);
```

The intersection of the line l and of the plane p , if it exists, is computed. If there is an intersection reduced to a point, the function returns `true`, otherwise `false`. The returned point is i (local index).

This function will be illustrated with a high school plane geometry problem: $ABCD$ is a tetrahedron such that $AB = 3$, $AC = 6$, $AD = 4.5$. I is the point of $[AB]$ such that $AI = 1$ and J is the

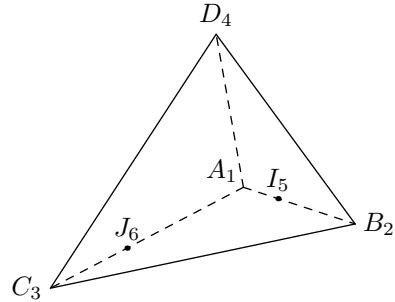


Figure 12: Tetrahedron: first construction.

point of $[AC]$ such that $AJ = 4$. We must determine the intersection of the line (IJ) with the plane (BCD) . We start by constructing the tetrahedron (figure 12).

We should first notice that several tetrahedra are fulfilling the requirements: B , C and D are independent. In order to obtain a rather general construction, we must parameterize it. A can for instance be set in $(0,0,0)$, B in $(3 \cos \beta, 3 \sin \beta, 0)$, C in $(6 \cos \gamma, 6 \sin \gamma, 0)$ and D obtained by the means of two rotations, one around \vec{k} , the other around a vector orthogonal to \vec{k} . In `METAPOST`, this is done with the commands given in figure 13.

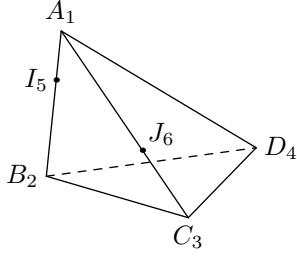
On figure 12, the numbers of the points have been added as indices. This figure is of course not well suited to this problem, because (BCD) is facing the observer. We will therefore move the observer, for instance to $C + 5\overrightarrow{DC} + 5\overrightarrow{BC} + 3\overrightarrow{CA}$ (figure 14). In order to achieve this shift, we have reached to tetrahedra points outside of the tetrahedron with the `pnt_obj` function (this function makes it also possible to define points of an object using points from another object):

```
new_vec(v_a);
new_vec(v_b);
new_vec(v_c);
vec_diff_(v_a,pnt_obj("tetra",3),
           pnt_obj("tetra",4)); %  $\overrightarrow{DC}$ 
vec_mult_(v_a,v_a,5); %  $5 \cdot \overrightarrow{DC}$ 
vec_diff_(v_b,pnt_obj("tetra",3),
           pnt_obj("tetra",2)); %  $\overrightarrow{BC}$ 
vec_mult_(v_b,v_b,5); %  $5 \cdot \overrightarrow{BC}$ 
vec_diff_(v_c,pnt_obj("tetra",1),
           pnt_obj("tetra",3)); %  $\overrightarrow{CA}$ 
vec_mult_(v_c,v_c,3); %  $3 \cdot \overrightarrow{CA}$ 
%  $C + 5\overrightarrow{DC}:$ 
vec_sum_(Obs,pnt_obj("tetra",3),v_a);
%  $C + 5 \cdot \overrightarrow{DC} + 5 \cdot \overrightarrow{BC}:$ 
vec_sum_(Obs,Obs,v_b);
%  $C + 5 \cdot \overrightarrow{DC} + 5 \cdot \overrightarrow{BC} + 3 \cdot \overrightarrow{CA}:$ 
vec_sum_(Obs,Obs,v_c);
free_vec(v_c);
free_vec(v_b);
free_vec(v_a);
```

```

set_point(1)(0,0,0); % A
set_point(2)(3*cosd(b),3*sind(b),0); % B
set_point(3)(6*cosd(c),6*sind(c),0); % C
new_vec(v_a); new_vec(v_b);
vec_def_vec_(v_a,vec_I); %  $\vec{v}_a \leftarrow \vec{r}$ 
vec_rotate_(v_a,vec_K,d); % rot. of  $\vec{v}_a$  around  $\vec{k}$  by an angle d
vec_prod_(v_b,v_a,vec_K); %  $\vec{v}_b \leftarrow \vec{v}_a \wedge \vec{k}$ 
vec_rotate_(v_a,v_b,e); % rot. of  $\vec{v}_a$  around  $\vec{v}_b$  by an angle e
vec_mult_(v_a,v_a,4.5);
vec_sum_(pnt(4),pnt(1),v_a); % D
free_vec(v_b); free_vec(v_a);
% Determination of I and J:
%  $I = A + \overrightarrow{AB}/\|\overrightarrow{AB}\|$ 
vec_diff(5,2,1); %  $\overrightarrow{V_5} \leftarrow \overrightarrow{AB}$ 
vec_unit(5,5); %  $\overrightarrow{V_5} \leftarrow \overrightarrow{AB}/\|\overrightarrow{AB}\|$ 
vec_sum(5,5,1); %  $I \leftarrow A + \overrightarrow{AB}/\|\overrightarrow{AB}\|$ 
%  $J = A + 4 \cdot \overrightarrow{AC}/\|\overrightarrow{AC}\|$ 
vec_diff(6,3,1); %  $\overrightarrow{V_6} \leftarrow \overrightarrow{AC}$ 
vec_unit(6,6); %  $\overrightarrow{V_6} \leftarrow \overrightarrow{AC}/\|\overrightarrow{AC}\|$ 
vec_mult(6,6,4); %  $\overrightarrow{V_6} \leftarrow 4 \cdot \overrightarrow{AC}/\|\overrightarrow{AC}\|$ 
vec_sum(6,6,1); %  $J \leftarrow A + 4 \cdot \overrightarrow{AC}/\|\overrightarrow{AC}\|$ 

```

Figure 13: Code for figure 12.**Figure 14:** Tetrahedron: second construction.

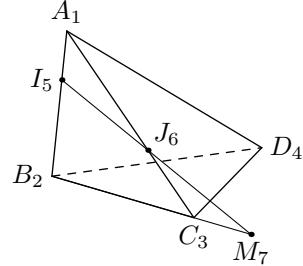
We now come to the computation of the intersection of the (IJ) line with the (BCD) plane (figure 15).

```

new_plane(bcd)(2,3,4);
new_line(ij)(5,6);
boolean b;
b:=def_inter_p_1_p1(7)(ij)(bcd):
if not b: message "no intersection"; fi;
free_line(ij);
free_plane(bcd);

```

Two other intersections can be computed using K , the middle of $[AD]$ (figure 16). The three intersections are then aligned. (We can see that L , M and N are as a matter of fact on the intersection of the planes (IJK) and (BCD) .) We have shown the alignment with a segment slightly extending on each side, using

**Figure 15:** Tetrahedron: third construction.

```
draw_line_extra(9,10)(-0.1,1.1);
```

The second pair of parameters indicates how much the segment extends on each side. $(0, 1)$ corresponds to no extension and a smaller (or larger) value for the first (or second) parameter produces an extension.

Figure 16 also illustrates the famous theorem by Girard Desargues (1639), which is the cornerstone of projective geometry. According to this theorem, two triangles BCD and IJK being given in the plane, the lines (BI) , (CJ) , (DK) have an intersection if and only if the intersections of (IK) and (BD) , of (IJ) and (BC) , and of (KJ) and (DC) are aligned. In our case, the lines (BI) , (CJ) , (DK) have indeed an intersection, namely A , and the intersections L ,

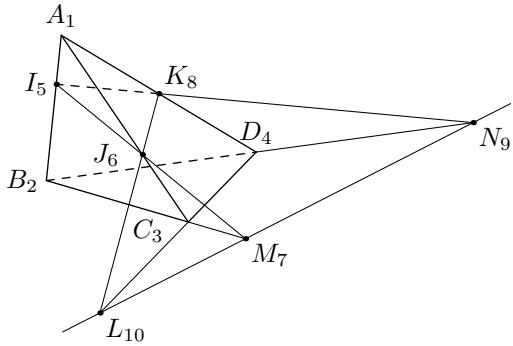


Figure 16: Tetrahedron: fourth construction.

M and N are aligned. The theorem states that the converse also holds. Seen in space, the theorem is very simple, but a purely plane proof is difficult.

Another function of 3dgeom allows for the direct computation of the intersection between two planes:

```
b:=def_inter_1_pl_p1(l)(p)(q);
```

where **b** is a **boolean**. If the intersection between the planes p and q is a line, the function yields **true** and the line is stored in l . Otherwise, the function yields **false**. Let us see on an example how this function can be used. Consider the drawing of a tetrahedron $SABC$ whose edges \widehat{SA} , \widehat{SB} et \widehat{SC} are known, as well as the angles \widehat{ASC} , \widehat{ASB} and \widehat{BSC} . Figure 17 shows such a tetrahedron with $SA = 9$, $SB = 8$, $SC = 4$, $\widehat{ASC} = 60^\circ$, $\widehat{ASB} = 40^\circ$ and $\widehat{BSC} = 30^\circ$. Contrary to the previous example, here we do not have a wide margin to place the points. We can of course start to construct the SAC triangle. It then only remains to place the B vertex.

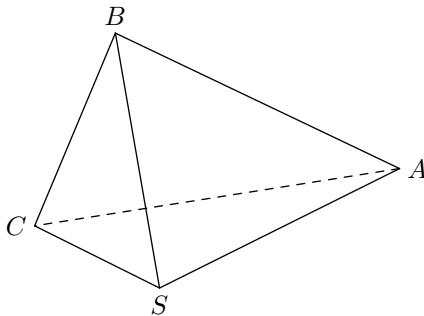


Figure 17: A tetrahedron specified by three lengths and three angles.

The angles \widehat{ASB} and \widehat{BSC} being given, B is obviously located on two cones: the cone of axis (SA) , of apex S and of angle \widehat{ASB} and the cone of axis (SC) , of apex S and of angle \widehat{CSB} . These two

cones in general intersect in two lines and B is on one of these intersections at the distance SB of S .

With a well-chosen implementation of a cone structure, the intersection can of course be automatically obtained. But it is also possible to use more restricted means by observing that we can draw the heights $[SH]$ and $[SK]$ stemming from B for each of the triangles SAB and SBC . For instance, $SH = SB \cdot \cos(\widehat{ASB})$. We can then define the planes orthogonal to the lines (SA) and (SC) going through the two heights' feet. The function

```
def_orth_pl_1_p(p)(l)(i);
```

constructs the plane p orthogonal to the line l and going through the local point i .

The intersection between the two constructed planes can then be computed. This intersection is a line orthogonal to the plane of the triangle SAC . On this line, we look for a point B at a given distance from S . The function call

```
b:=def_point_at(i)(d)(j)(1);
```

where **b** is a **boolean** variable, defines the local point i as being a point of the line l at a distance $|d|$ from the local point j if such a point can be found. In that case, the return value of the function is **true**, otherwise it is **false**. In general, two points satisfy the condition and the function will return either one depending on the sign of d .

Essentially, the figure is thus produced by the commands in figure 18.

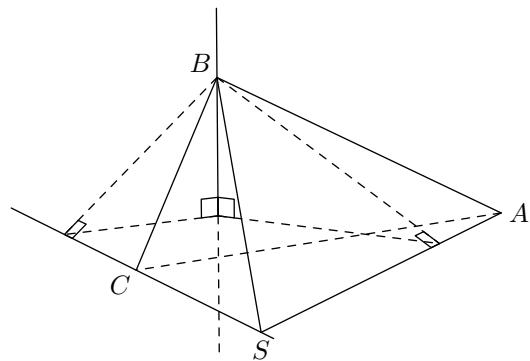


Figure 19: A tetrahedron specified by three lengths and three angles (construction).

The whole construction is given in figure 19. In order to draw it, we have “unlocalized” points such as the heights’ feet and the intersection between the perpendicular in B to the (SAC) plane. In order to produce the right angles, we have used the commands **def_right_angle** for the definition and **draw_double_right_angle** for the drawing. Each

```

new_point(h); new_point(k);
set_point(1)(0,0,0); % S
set_point(2)(lsa,0,0); % A
set_point(4)(lsc*cosd(aasc),lsc*sind(aasc),0); % C
vec_diff_(h,pnt(2),pnt(1));
vec_unit_(h,h);
vec_mult_(h,h,lsb*cosd(aasb));
vec_sum_(h,h,pnt(1)); % H
vec_diff_(k,pnt(4),pnt(1));
vec_unit_(k,k);
vec_mult_(k,k,lsb*cosd(absc));
vec_sum_(k,k,pnt(1)); % K
new_plane(hp)(1,1,1); % initialization to three points
new_plane(kp)(1,1,1); % ditto
new_line(sa)(1,2); % (SA)
new_line(sc)(1,4); % (SC)
new_line(inter)(1,1); % intersection line of the two planes
def_orth_pl_l_p_(hp)(sa)(h); % plane orthogonal to (SA) in H
def_orth_pl_l_p_(kp)(sc)(k); % plane orthogonal to (SC) in K
if def_inter_l_pl(inter)(hp)(kp): % there is an intersection
    if not def_point_at(3)(-lsb,1)(inter): % B
        message "Should not happen";
    fi;
else:
    message "PROBLEM (probably the angle ASC too small)";
    set_point(3)(1,1,1);
fi;
free_line(inter); free_line(sc); free_line(sa);
free_plane(kp); free_plane(hp);
free_point(k); free_point(h);

```

Figure 18: Code for figure 19.

right angle is made of two segments, defined using three points. These three points are new object points. For instance, one of the right angles is created with

```
def_right_angle(7,8,9,5,1,3);
```

This means that three points (numbered locally 7, 8, and 9) are introduced and that they are set according to the angle determined by the triangle of points (5,1,3).

The drawing, on the other hand, is simpler:

```
draw_double_right_angle(7,8,9,5);
```

4.7 Visual complements

4.7.1 Representation of planes

A plane is often represented using four particular points making a rectangle. These points must be defined. The drawing of an horizontal rectangle corresponding to the (SAC) plane in figure 19 can be obtained as follows (see figure 20):

```

set_point(14)(-2,-2,0); % p1
set_point(15)(11,-2,0); % p2
set_point(16)(11,10,0); % p3
set_point(17)(-2,10,0); % p4

```

The points are connected with `draw_lines`.

4.7.2 Hidden parts and visual intersections

As shown on figure 20, there is (currently) no automatic hidden parts removal or a special treatment of those parts. It is necessary to handle the dashed lines by hand and this is only practical in the case of non-moving images. With animations, the perspective can be subjected to such variations that it is advisable to have an automatic solution of the problem.

However, on a non-moving image, the problem of removing (or handling in a special way) hidden parts is rather simple to express and solve. It is actually a matter of determining “apparent” intersections between two curves. In the previous example,

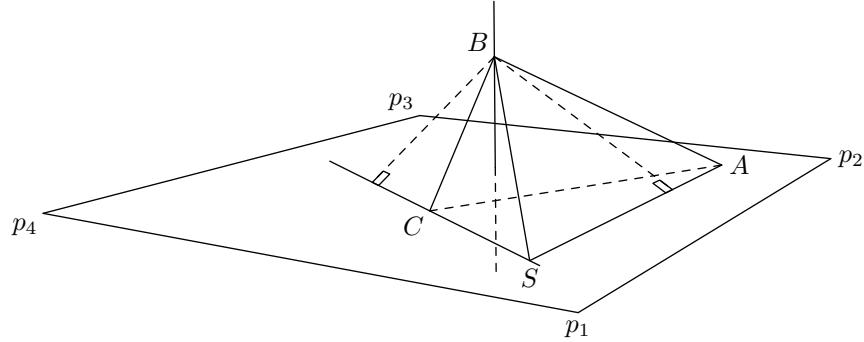


Figure 20: The addition of a plane to the drawing in figure 19.

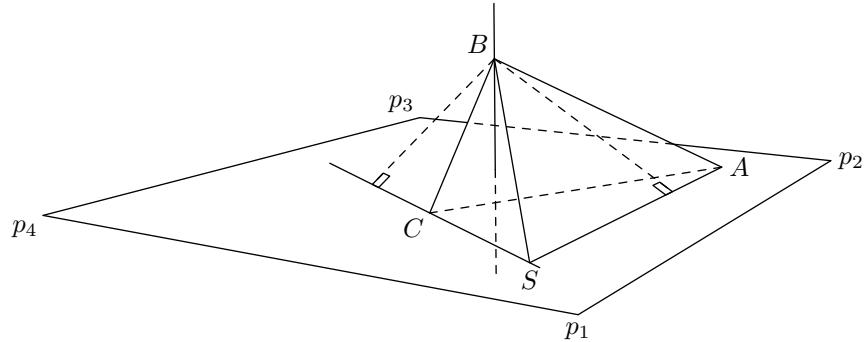


Figure 21: The interruption of a plane.

sides of the tetrahedron seem to meet sides of the rectangle representing the plane, even though they do not meet in space.

If the plane had to be represented in a clearer way, the apparent (or visual) intersections of the sides $[BA]$ and $[BC]$ with the most distant side of the rectangle would have to be determined. How can this be done?

One way is to construct a point of the segment $[p_2p_3]$, but as a function of the observer's position. The intersection between (Obs, p_2, p_3) and (Obs, A, B) is a line going through the observer and through the point of (p_2p_3) of interest to us. It is then sufficient to determine the intersection between this line and the (p_2p_3) line. This intersection being computed in space, care must be taken that rounding errors can prevent the intersection of two lines which should otherwise intersect. The `def_inter_p_1_1` function finds the middle of the two points of each line where the line is closest to the other line. This function also returns the distance between the two points. This makes it possible to find a point of (p_2p_3) corresponding to the vi-

sual interruption caused by the $[BA]$ segment. Similarly, it is possible to find the point corresponding to the visual interruption caused by the $[BC]$ segment. These two points, with p_2 and p_3 , make it possible to draw a more natural plane. This is what is done in figure 21.

The `def_visual_inter` function takes care of this procedure. It takes four local points and computes a fifth one:

```
boolean b;
b:=def_visual_inter(i)(j,k,l,m);
```

If the function returns `true`, point i is located on (jk) at the apparent intersection of (jk) and (lm) .

It should be noted that in a central perspective, the computed intersections depend on the observer's position. If this position changes, the intersections must be recomputed. As a consequence, as we already indicated it, it is necessary to call the `reset_obj` function after the redefinition of the observer's position.

In a parallel projection, the observer is not used in the computation of the visual intersection, but the interface remains the same. The intersections

are almost computed in the same way, except that the planes whose intersection is computed are not determined by the observer and a segment, but by the projection direction and a segment.

4.7.3 Vanishing points

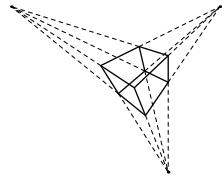


Figure 22: The three classical vanishing points of a cube.

The representation of an object containing parallel segments in a central perspective shows vanishing lines and points. The projected lines are usually no longer parallel and intersect. An example of a cube representation with three vanishing points is given in figure 22. The classical representations often distinguish the drawings with one, two or three vanishing points. The drawings with one or two vanishing points are special cases corresponding to lines which are parallel to the projection plane. In figure 22, none of the cube's sides are parallel to the projection plane, and this leads to vanishing points. If the projection plane had been parallel to the vertical segments of the cube, these segments wouldn't have exhibited vanishing points. If it is a whole face which is parallel to the projection plane, there is only one vanishing point left.

The vanishing points correspond to points located at the infinite in space, along a direction going through the observer and directed by a vector corresponding to the object's segment. Very often, some of the vanishing points will be quite distant on the drawing, and possibly outside the drawing.

The classical representations in architecture or in painting put two vanishing points on an horizontal line and a third vanishing point corresponding to the vertical vanishing lines. Figure 22 differs from that representation because the observer is not oriented along a vertical axis.

A different number of vanishing points do not correspond to different projections, but on the one hand to objects which are positioned differently in space with respect to the projection plane, and on the other hand to objects of different nature. A sphere will of course have no vanishing point! The number of vanishing points can actually be any number, including a number greater than three. It suffices to choose a pair of parallel lines on the ob-

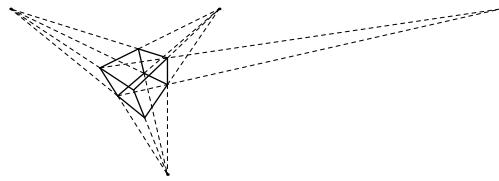


Figure 23: A fourth vanishing point.

ject. Figure 23 shows that besides the three classical vanishing points, the six vanishing points stemming from the diagonal segments of the cube can be considered. One of those is represented on the figure. (It should be remarked that vanishing points being very sensitive to the location of the observer, it is rather difficult to find by hand a location where the nine vanishing points are simultaneously visible in a restricted space.) More complex objects would have even more vanishing points.

A vanishing point can be determined in a simple way by computing the intersection between the projection plane and the line going through the observer and oriented by a vector of the object. The following lines find the vanishing point of the segment connecting two points numbered 1 and 5:

```
% defines the projection plane
def_screen_pl(screen);
new_line(1)(1,5);
if not
  def_vanishing_point_p_1_pl(11)(1)(screen):
    message "no vanishing point";
    set_point(11)(0,0,0);
fi;
...
```

Like for the visual intersections, the vanishing points depend on the observer's position and each time the point of view changes (either because the observer moves, or because the object moves), the object points must be recomputed with `reset_obj`.

It would also be possible to recompute the vanishing points directly in the plane. This would be a mere application of `whatever` (see section 2).

4.7.4 Shadows

The shadows corresponding to projections are simulated by shading the projected part. An example is given in figure 24. In that example, a triangle is projected on a plane. We have merely computed the projections of the three vertices of the triangle and we have then shaded the projection. This technique works for each projection, as long as the projection is made along a line and on a plane or a set of planes.

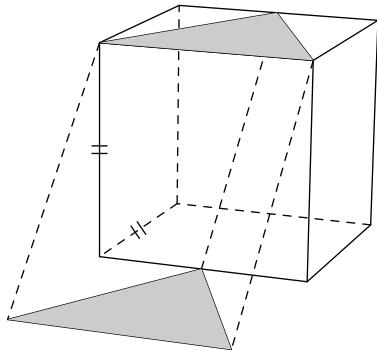


Figure 24: The cube with a shadow.

5 Modifications with respect to the first distribution

When we worked on the new version of the 3d package, we made various changes which seemed to go in the direction of a greater homogeneity. A number of elementary functions have been renamed in order to respect our new function naming conventions. Hence, our first article (Roegel, 1997) on that matter is now no longer strictly correct because functions such as `vect_sum` must be replaced by `vec_sum`. All the “`vect`” have been replaced by “`vec`.” Finally, the creation of vectors and points has slightly changed. The differences are not especially important, but in order to make the transition easier for the reader, we have put on CTAN a corrected version of the original article, with differences highlighted.

6 Conclusion and limits

This package contains numerous features which have not been mentioned and it is easy to add new ones, in particular concerning the manipulation of new structures. We silently omitted drawing curves such as circles which involve another module that will be described elsewhere. More comprehensive documentation comes with the package.

A study of this article also reveals that the coordinates of a point have only explicitly been used for a few points constructed directly with `set_point` or `set_point`. All other points have been obtained through various geometric operations. It doesn’t mean that the coordinates are not accessible! They are given by the functions `xval`, `yval` and `zval` applied to a vector or point reference.

This package is of course not perfect and has limits. Besides METAPOST’s limits, in particular in numerical capacity (the dimensions are bounded to 4096 PostScript points, at least for a normal use), our package lacks automatic computations. Still

many manual interventions are needed. Other limitations concern (currently) the absence of a decent and automatic hidden parts removal algorithm. Finally, error handling in METAPOST will not be simple for who is not somewhat accustomed with the language.

This work can of course be compared to other works going in the same direction. First, it should be clear that we do not claim to compete with professional CAD or computer algebra tools. We want above all to provide a light and powerful system helping the creation of geometric constructions, in particular suited for a geometry class in high school. In the `TeX` world, there are to our knowledge only few works integrating space. METAGRAF (<http://w3.mecanica.upm.es/metapost>), also based on METAPOST, is an interactive system with a notion of space, but which doesn’t seem to provide possibilities of geometric constructions, animations, changes of perspective, etc. The PSTricks system has a 3D module, but it is relatively undeveloped. The computations are done with `TeX` and extending the system is tedious. Outside the `TeX` world, various 3D languages are available, in particular OpenGL, which goes much beyond our system with respect to rendering. As a teaching tool for geometry, we should also mention the *Cabri-Géomètre* software (cf. <http://www.cabri.net>).

7 Acknowledgements

I would like to thank Nicolas Kisselhoff who led me to develop the `3dgeom` module by providing me with figures from his geometry course, Sami Alex Zaimi who developed the notion of an object and has indirectly influenced this work and Pablo Argon who convinced me many years ago of the usefulness of METAPOST for the geometry of the ruler and the compass. Hans Hagen proofread the French version of the article and pushed me to clarify it even further, in particular through the introduction of new figures. Finally, Jean-Michel Huffen and Damien Wyart have made corrections and suggested various improvements.

References

- Dürer, Albrecht. *Underweysung der messung mit dem zirckel und richtscheyt in Linien ebnen unnd gantzen corporen durch Albrecht Dürer zu samen getzogen und zu nutz aller kunstliebhabenden mit zu gehörigen figuren in truck gebracht im jar. M.D.X.X. V. 1525*. Facsimile (Portland, Or.: Collegium Graphicum, c1972.).

- Goossens, Michel, S. Rahtz, and F. Mittelbach. *The L^AT_EX Graphics Companion*. Reading, MA, USA : Addison-Wesley, 1997.
- Gourret, Jean-Paul. *Modélisation d'images fixes et animées*. Paris : Masson, 1994.
- Hagen, Hans. *MetaFun*, 2002. <http://www pragma-ade.com>.
- Hobby, John D. "A User's Manual for MetaPost". Technical Report 162, AT&T Bell Laboratories, Murray Hill, New Jersey, 1992. <http://cm bell-labs.com/who/hobby/MetaPost.html>.
- Hoenig, Alan. *T_EX unbound. L^AT_EX & T_EX Strategies for Fonts, Graphics, & More*. Oxford, New York : Oxford University Press, 1998.
- Knuth, Donald E. *Computers & Typesetting, volume C: The METAFONTbook*. Reading, MA : Addison-Wesley Publishing Company, 1986.
- Krikke, Jan. "Axonometry: A Matter of Perspective". *IEEE Computer Graphics and Applications* **20**(4), 7–11, 2000. <http://www.computer.org/cga/cg2000/pdf/g4007.pdf>.
- Le Goff, Jean-Pierre. "De la perspective à l'infini géométrique". *Pour la Science* (278), 66–72, 2000.
- Meyer, Bertrand. *Object-oriented software construction*. Upper Saddle River, N.J.: Prentice Hall, 1997.
- Roegel, Denis. "Creating 3D animations with METAPOST". *TUGboat* **18**(4), 274–283, 1997. <ctan:graphics/metapost/macros/3d/tugboat/tb57roeg.pdf>. An updated version is located at the same place.

◊ Denis Roegel
LORIA
BP 239
54506 Vandœuvre-lès-Nancy
FRANCE
roegel@loria.fr
<http://www.loria.fr/~roegel>

The plot functions of ***pst-plot****

Jana Voß and Herbert Voß

Abstract

Plotting of external data records is one of the standard problems of technical and industrial publications. One common approach is importing the data files into an external program, such as *gnuplot*, provided with axes of coordinates and further references, and finally exported to *LATEX*. By contrast, in this article we explain ways to get proper data plotting without using external applications.

1 Introduction

The history and the meaning of PostScript have been covered sufficiently in many articles. For the programming language PostScript have a look at [3, 4]. The package *pst-plot* [8] under consideration here is part of the *pstricks* project. It must be loaded into a *LATEX* document as usual with `\usepackage{pst-plot}` or alternatively, for documents written in plain *TeX*, with `\input pst-plot.tex`.

pst-plot provides three plot macros for the representation of external data, with the following syntax:

```
\listplot* [<parameter>] {<data macro>}
\dataplot* [<parameter>] {<data macro>}
\fileplot* [<parameter>] {<file name>}
```

The starred forms have the same meaning as with all macros of *pstricks*: to plot the data in a reversed mode. Thus, for a default black-on-white diagram one produces the negative with the starred command form, namely white-on-black. Additionally, a negative plot implies that PostScript closes the path of the points from the last to the first one and fills all points inside this closed path with the actual *fillcolor*. For our purposes in this article there will be no real sense in this negative view; therefore, all of the following examples are plotted with the normal (unstarred) form only.

For further information about *pstricks*, have a look at the (more or less) official documentation [6], or the extensive description in the “standard *LATEX*” book [2] or in [1, 9]. Altogether, however, these do not fully describe the substantial differences between these three plot macros.

For all of the examples in this article, the complete *pspicture* environment is indicated, so that the examples may be directly copied. The documentation for the *multido* macro used here can be found in the package itself [7]; macros not otherwise mentioned are described in the *pstricks* documentation [6].

* Based on “Die Plot-Funktionen von *pst-plot*,” *Die TeXnische Komödie 2/02*, June 2002, pages 27-34, with permission.

Table 1: Possible options

style option	meaning
plotstyle=dots	plot (x,y) as a dot
=line	draw a line from a dot to the following one
=polygon	nearly the same as line, but with a line from the last to the first dot
=curve	interpolation between three dots, whereby the curve can go beyond the point of origin and/or termination point
=ecurve	like curve, but ends at the first/last dot
=ccurve	like curve, but closed

The general `plotstyle` parameter is particularly important, and can take the values shown in table 1.

By default, the `line` option is selected.

The following general commands are also useful in conjunction with the plot commands. They are also defined by the `pst-plot` package:

```
\readdata{<data macro name>}{|<file name>}
\savedata{<data macro name>}{|<file name>}
```

In the following examples only the `\readdata` macro is used, but it would be straightforward to create examples with `\savedata`.

2 Examples for `\listplot`

The syntax of `\listplot` is :

```
\listplot{|<data macro name>}
```

The data macro may contain any additional (L^AT_EX-) or PostScript-commands. The (L^AT_EX-) macros are expanded first before they are passed as a **list** of $x|y$ values to PostScript. The data records can be defined inside the document like

```
\newcommand{\dataA}{%
 1.00000000 1.00000000
 0.56000000 0.31000000
 0.85841600 0.17360000
 ...}
```

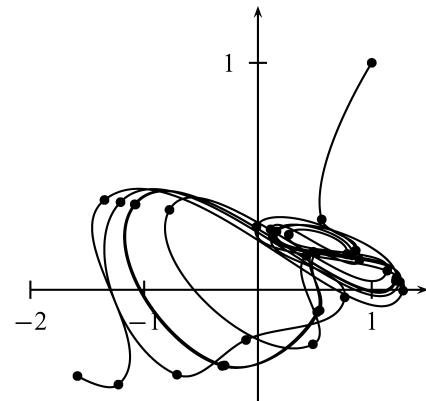
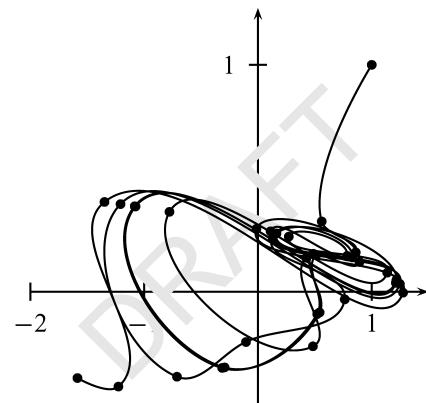
or can be read from an external data file with `\readdata`:

```
\readdata{\dataA}{/anyPath/data.dat}
```

The example in figure 1 shows the Henon attractor, a typical graphic of a system with chaotic behavior[5].

Figure 2 is nearly the same as figure 1, with the addition of PostScript code to get the "Draft" watermark. (Some familiarity with PostScript is needed to fully understand its operation.) To save space, listing 2 does not contain the data, which is nothing more than a sequence

of pairs of floats, each value separated by a space, as shown above.

Figure 1: Example for `\listplot`Figure 2: Example for modified `\listplot`Listing 1: L^AT_EX source for figure 1

```
1 \readdata{\henon}{henon.dat}
2 \psset{xunit=1.5cm, yunit=3cm}
3 \begin{pspicture}(-3,-0.5)(2.25,1.25)
4 \psaxes{>}(0,0)(-2,-0.5)(1.5,1.25)
5 \listplot[%
6   showpoints=true,%
7   linecolor=red,%
8   plotstyle=curve]{\dataA}
9 \end{pspicture}
```

Listing 2: L^AT_EX source for figure 2

```
1 \newcommand{\DataA}{%
2   [ ... data ... ]
3   gsave           % save graphic status
4   /Helvetica findfont 40 scalefont setfont
```

```

5   45 rotate      % rotate by 45 degrees
6   0.9 setgray    % 1 is color white
7   -60 10 moveto (DRAFT) show
8   grestore
9 }
10 \psset{xunit=1.5cm, yunit=3cm}
11 \begin{pspicture}(-3,-0.5)(2.25,1.25)
12   \psaxes{->}(0,0)(-2,-0.5)(1.5,1.25)
13   \listplot[%
14     showpoints=true,%
15     plotstyle=curve]{\dataA}
16 \end{pspicture}

```

Naturally, [...] data [...] is replaced by all the $x|y$ -values; they're omitted here only to save space.

As an alternative to direct modification of the data set passed to `\listplot`, one can redefine the macro `defScalePoints` from `pst-plot`. For example, to change the $x|y$ values and then rotate the whole plotted graphic (don't ask why!), the redefinition is as shown in listing 3.

Listing 3: `LATEX` source for figure 3

```

1 \makeatletter
2 \pst@def{ScalePoints}<%
3 %-----
4   45 rotate % rotate the whole object
5 %
6   /y ED /x ED
7   counttomark dup dup cvi eq not { exch pop
8     } if
9   /m exch def /n m 2 div cvi def
10  n {
11    %-----
12    exch % exchange the last two elements
13    %-----
14    y mul m 1 roll
15    x mul m 1 roll
16    /m m 2 sub
17    def } repeat>
18 \makeatother

```

This gives figure 3.

Thus, the advantage of `\listplot` is that one can easily modify the data values without any external program. Here is one more example—suppose you have the following data records:

```

1050, 0.368
1100, 0.371
1200, 0.471
1250, 0.428
1300, 0.391
1350, 0.456
1400, 0.499
1500, 1.712
1550, 0.475
1600, 0.497

```

which perhaps came automatically from a technical device. The unit of the x-values is micrometer but it makes

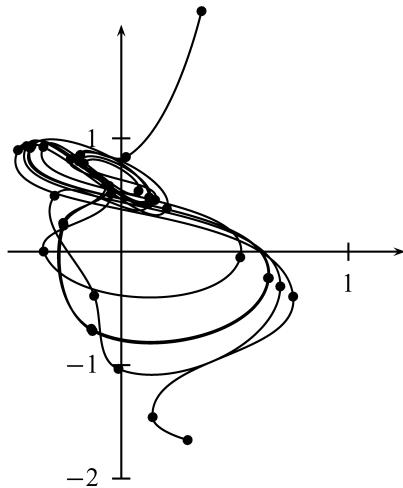


Figure 3: Example for `\listplot` with a redefined `ScalePoints`

more sense to use millimeter for the plot. A redefinition of `ScalePoints` makes it very easy to plot the data with this change of scale:

Listing 4: Rescale all x values

```

1 \makeatletter
2 \pst@def{ScalePoints}<%
3 /y ED /x ED
4 counttomark dup dup cvi eq not { exch pop
5   } if
6 /m exch def /n m 2 div cvi def
7 n {
8   y mul m 1 roll
9   x mul 1000 div m 1 roll% --- divide by
10  1000
11 /m m 2 sub
12 def } repeat>
13 \makeatother

```

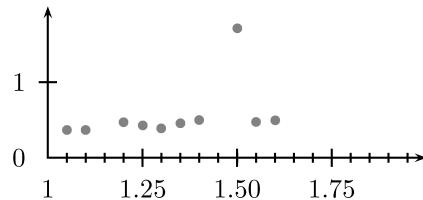


Figure 4: Example for modified data values with a redefined `ScalePoints`

3 Examples for `\dataplot`

`\dataplot` has the same syntax as `\listplot`, so the first question is, what is the difference between the two?

\listplot builds a list of all the data and then multiplies all values with the length unit. This takes some time, so you may prefer a so-called “quick plot”, where the data can be passed more quickly to PostScript, depending on the plotstyle and especially the option showpoints. Table 2 shows whether this is possible. A quick plot is not possible with \listplot, whereas \dataplot uses it whenever possible. When it is not possible, \dataplot simply calls \listplot.

Table 2: Possible options for a ”quick plot“

plotstyle	options	macro
line	all, except linearic, showpoints, arrows,	quick plot \listplot
polygon	all, except linearic, showpoints	quick plot \listplot
dots	all	quick plot
bezier	all, except arrows, showpoints	quick plot \listplot
cbezier	all, except showpoints	quick plot \listplot
curve	all	quick plot \listplot
ecurve	all	quick plot \listplot
ccurve	all	quick plot \listplot

\dataplot needs to be passed a macro holding the data. The data is typically saved in an external file, which can be read by (for instance) the \readdata macro, as follows:

```
\readdata{<object name>}{<data file>}
```

For example:

```
\readdata{\feigenbaum}{feigenbaum.data}
```

The amount of data is limited only by TeX’s memory. The above example can be plotted with:

```
\dataplot{\feigenbaum}
```

Overlays with different data files are also possible. For example, figure 5 shows the use of two different data files which are plotted using one coordinate system. It shows the sorting time for “Bubble-Sort” and “Select-Sort” as a function of the number of the elements.

Listing 5: L^AT_EX source for figure 5

```

1  \psset{xunit=0.0005cm,yunit=0.005cm}
2  \begin{pspicture}(0,-50)(10000,1100)
3    \readdata{\bubble}{bubble.data}
4    \readdata{\select}{select.data}
5    \dataplot[%]
6      plotstyle=line,%
7      linecolor=blue]{\bubble}
8    \dataplot[%]
9      plotstyle=line,%
10     linecolor=red,%

```

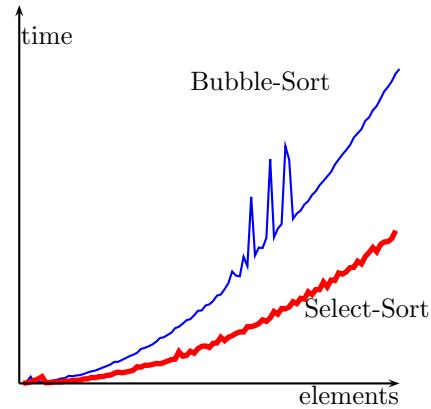


Figure 5: Example for \dataplot

```

11   linewidth=2pt]{\select}
12   \psline{->}(0,0)(10000,0)
13   \psline{->}(0,0)(0,1000)
14   \rput[1](20,995){time}
15   \rput[r](9990,-20){elements}
16   \rput[1](4500,800){Bubble-Sort}
17   \rput[1](7500,200){Select-Sort}
18 \end{pspicture}

```

In short, the advantage of \dataplot is the possibility of a ”quick plot“, and the advantage of \listplot is that it is easy to manipulate the data values before they are plotted.

4 Examples for \fileplot

\fileplot can be used whenever (*x|y*) data that is saved in a file is to be plotted. The values must be given as pure numerical values in pairs, one pair on each line, and may have spaces, commas, parentheses, and braces as punctuation, as follows:

```
x y
x,y
(x,y)
{x,y}
```

The tab character (\t or ASCII \009) is often used as a separator, but tab is *not* valid here. Tabs may be converted to spaces in many ways, for example with the standard Unix utility tr:

```
tr '\t' ' ' <inFile >outFile
```

The data files may also contain % characters, but no other characters are allowed.

Our first example for \fileplot is shown in figure 6, which is an UV/VIS absorber spectrum $A = \lg \frac{I_0}{I}$ as a function of the wavelength. The second example (figure 7) shows the evolution of a population as a function of the spawn factor (Feigenbaum diagram [5]). The source code for these images is shown in listings 6 and 7.

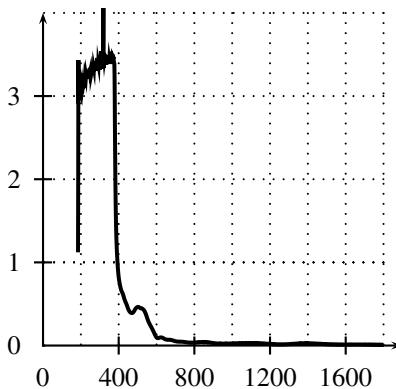


Figure 6: Example for \fileplot

Listing 6: L^AT_EX source for figure 6

```

1  \psset{xunit=0.0025cm,yunit=1.1cm}
2  \begin{pspicture}(-25,-.25)(1950,4)
3  \fileplot[plotstyle=line]{fileplot.data}
4  \psaxes[dx=400,Dx=400]{->}(1900,4)
5  \multido{\n=200+200}{9}{%
6    \psline[linestyle=dotted](\n,0)(\n,4)%
7  }
8  \multido{\n=+1}{5}{%
9    \psline[linestyle=dotted]%
10   (0,\n)(1800,\n)%
11 }
12 \end{pspicture}

```

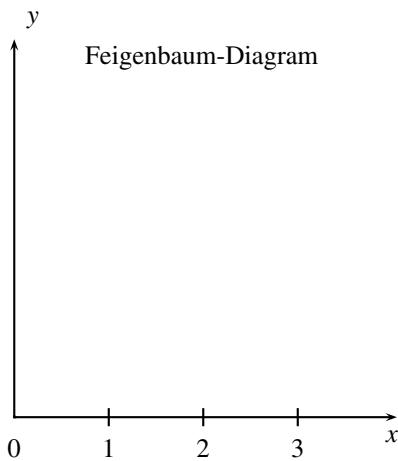


Figure 7: Another example for \fileplot

Listing 7: L^AT_EX source for figure 7

```

1  \psset{xunit=1.5cm,yunit=6cm}
2  \begin{pspicture}(-0.25,-0.05)(4.25,1)
3  \fileplot[plotstyle=dots]{%
4    feigenbaum.data}

```

```

5  \psaxes{->}(0,0)(4.05,1)
6  \rput(4,-0.05){$x$}
7  \rput(0.2,1.05){$y$}
8  \rput[1](0.2,3.75){Feigenbaum-Diagram}
9  \end{pspicture}

```

As you may see in listing 8, \fileplot does little of its own. It first calls \readdata to read the data, and then, depending on the kind of data and specified options, \fileplot uses \dataplot for a quick plot if possible. Otherwise, it falls back to \listplot.

Listing 8: The source of the \fileplot macro

```

1  \def\fileplot{\def\pst@par{}\pst@object{%
2    fileplot}}
3  \def\fileplot@#1{%
4    \pst@killglue
5    \begingroup
6      \use@par
7      \pst@false
8      \nameuse{testqp@\psplotstyle}%
9      \if@pst
10        \dataplot@ii{\pst@readfile{#1}}%
11      \else
12        \listplot@ii{\pst@altreadfile{#1}}%
13      \fi
14    \endgroup
15    \ignorespaces%
}

```

\fileplot has the advantage of being easy to use, but the disadvantage of needing a lot of memory: T_EX has to read all the data values before it can process anything. As a rule of thumb, when there are more than 1000 data entries T_EX's main memory must be increased. Furthermore, the running time may be enormous, especially on slow machines.

To prevent such problems, one can use the macro \PSTtoEPS to create an eps file. For more information, see the documentation of pstricks [6].

References

- [1] Denis Girou. Présentation de PSTRicks. *Cahier GUTenberg*, 16:21–70, April 1994.
- [2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The L^AT_EX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Mass., 1997.
- [3] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. IWT, Vaterstetten, 1989.
- [4] Glenn C. Reid. *Thinking in PostScript*. Addison-Wesley, Boston, 1990.
- [5] Herbert Voß. *Chaos und Fraktale selbst programmieren: von Mandelbrotmengen über Farbmanipulationen zur perfekten Darstellung*. Franzis Verlag, Poing, 1994.

- [6] Timothy van Zandt. *PSTricks - PostScript macros for generic TeX*. <http://www.tug.org/application/PSTricks>, 1993.
- [7] Timothy van Zandt. *multido.tex - a loop macro, that supports fixed-point addition*. CTAN:/graphics/pstricks/generic/multido.tex, 1997.
- [8] Timothy van Zandt. *pst-plot: Plotting two dimensional functions and data*. CTAN:graphics/pstricks/generic/pst-plot.tex, 1999.
- [9] Timothy van Zandt and Denis Girou. Inside PSTricks. *TUGboat*, 15:239–246, September 1994.

◊ Jana Voß
Wasgenstr. 21
14129 Berlin GERMANY
Jana@perce.de

◊ Herbert Voß
Wasgenstr. 21
14129 Berlin GERMANY
voss@perce.de
<http://www.perce.de>

plots. They are always used with the globally defined options

```
\psset{subgriddiv=0,griddots=5,%
       gridlabels=7pt}
```

2 The parallel projection

Figure 1 shows a point $P(x, y, z)$ in a three dimensional cartesian coordinate system (x, y, z) with a transformation into $P^*(x^*, y^*)$, the point in the two dimensional system (x_E, y_E) .

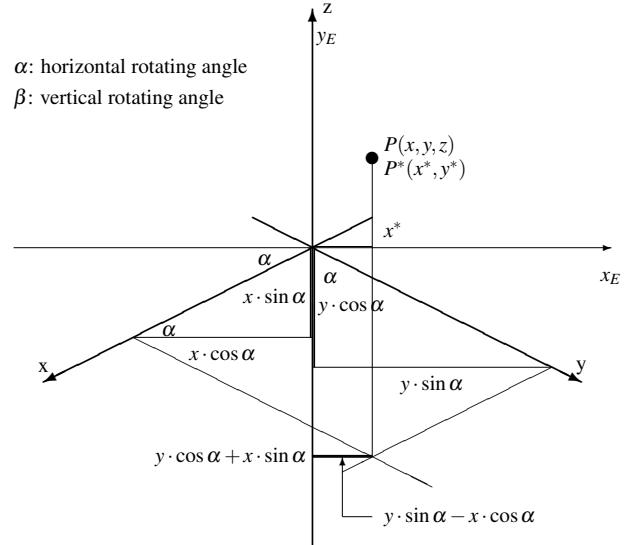


Figure 1: Lengths in a three dimensional system

Three dimensional plots with `pst-3dplot`

Herbert Voß

Abstract

The well-known `pstricks` package [7] offers excellent macros for creating more or less complex graphics which could be inserted into the document without having it exported to EPS or PDF. `pstricks` itself is the base for several other additional packages, which are typically named `pst-xxxx`, such as `pst-3dplot`.

There exist several packages for plotting three dimensional graphical objects. `pst-3dplot` handles three dimensional objects, mathematical functions, and data files similarly to `pst-plot` in two dimensions.

1 Introduction

The `pstricks` packages are available as usual from any possible CTAN server. The base parts are located at `CTAN:graphics/pstricks/generic/` and most of the additional packages at `CTAN:graphics/pstricks/contrib/` [7].

All `\psgrid` commands are only for a better view of the examples, they are not really necessary for the 3D-

The angle α is the horizontal rotation with positive values for anti-clockwise rotations of the 3D coordinates. The angle β is the vertical rotation (orthogonal to the paper plane). In figure 2 we have $\alpha = \beta = 0$. The y -axis comes perpendicularly out of the paper plane. Figure 3 shows the same for another angle with a view from the side, where the x -axis shows into the paper plane and the angle β is greater than 0 degrees.

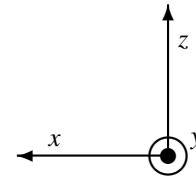


Figure 2: Coordinate system for $\alpha = \beta = 0$ (y -axis comes out of the paper plane)

The two dimensional x coordinate x^* is the difference of the two horizontal lengths $y \cdot \sin \alpha$ and $x \cdot \cos \alpha$ (figure 1):

$$x^* = -x \cdot \cos \alpha + y \cdot \sin \alpha \quad (1)$$

The z -coordinate is unimportant, because the rotation comes out of the paper plane, so we have only a

different y^* value for the two dimensional coordinate but no other x^* value. The β angle is well seen in figure 3 which derives from figure 2, if the coordinate system is rotated by 90 deg horizontally to the left and vertically by β also to the left.

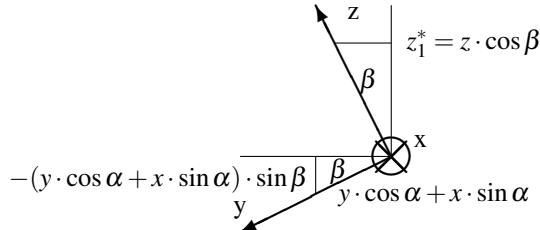


Figure 3: Coordinate system for $\alpha = 0$ and $\beta > 0$ (x -axis goes into the paper plane)

The value of the perpendicular projected z coordinate is $z^* = z \cdot \cos \beta$. With figure 3 we see that the point $P(x, y, z)$ runs on an elliptical curve when β is constant and α changes continuously. The vertical alteration of P is the difference of the two “perpendicular” lines $y \cdot \cos \alpha$ and $x \cdot \sin \alpha$. These lines are rotated by the angle β , so we have to multiply them with $\sin \beta$ to get the vertical part. We get the following transformation equations:

$$\begin{aligned} x_E &= -x \cos \alpha + y \sin \alpha \\ y_E &= -(x \sin \alpha + y \cos \alpha) \cdot \sin \beta + z \cos \beta \end{aligned} \quad (2)$$

or the same written in matrix form:

$$\begin{pmatrix} x_E \\ y_E \end{pmatrix} = \begin{pmatrix} -\cos \alpha & \sin \alpha & 0 \\ -\sin \alpha \sin \beta & -\cos \alpha \sin \beta & \cos \beta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (3)$$

3 Coordinate axes

The syntax for drawing the coordinate axes is

```
\pstThreeDCoor[<options>]
```

Without any options, we get the default view seen in figure 4 with the predefined values:

```
xMin=-1, xMax=4,
yMin=-1, yMax=4,
zMin=-1, zMax=4,
Alpha=45, Beta=30
```

There are no restrictions for the angles and the max and min values for the axes; all `pstricks` options are possible as well. The following example (5) changes the color and the width of the axes. The angles `Alpha` and `Beta` are important to all macros and should always be set with `psset` to make them global to all other macros. Otherwise they are only local inside the macro to which they are passed.

```
1 \begin{pspicture}(-2,-1)(1,2.25)
2     \psgrid
```

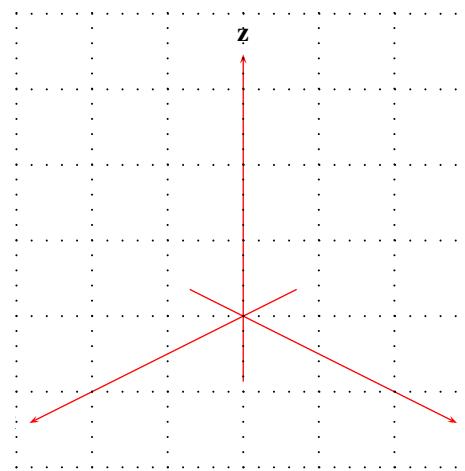


Figure 4: The default 3D coordinate system

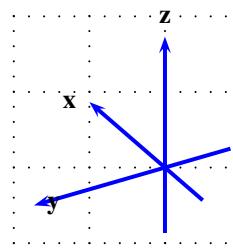


Figure 5: Axes with a different view and color

```
3 \psset{ Alpha=-60,Beta=30}
4 \pstThreeDCoor[%]
5     linewidth=1.5pt, linecolor=blue, %
6         xMin=-1, xMax=2, yMin=-1, yMax=2, %
7         zMin=-1, zMax=2]
8 \end{pspicture}
```

4 put command

The syntax is similar to the `\rput` macro from the package `pst-plot`:

```
\pstThreeDPut[<options>]%
    (x,y,z) {<any material>}
```

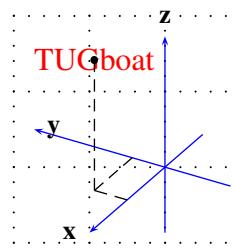


Figure 6: Example for the `\pstThreeDPut` macro

```
1 \begin{pspicture}(-2,-1)(1,2.25)
```

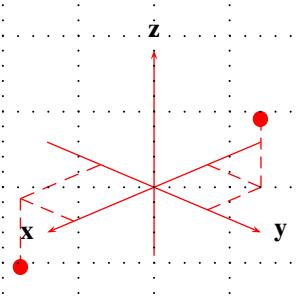


Figure 7: 3D dots with marked coordinates

```

2 \psgrid
3 \psset{ Alpha=-60,Beta=-30}
4 \pstThreeDCoor[%
5   linecolor=blue,%
6   xMin=-1,xMax=2,
7   yMin=-1,yMax=2,% 
8   zMin=-1,zMax=2]
9 \pstThreeDDot(1,0.5,2){\red{\large TUGboat}}
10 \pstThreeDDot[drawCoor=true](1,0.5,2)
11 \end{pspicture}

```

Internally, the `\pstThreeDPut` macro defines a two dimensional node `temp@pstNode` and then uses the default `\rput` macro from `pstricks`. Because of the perspective from which the coordinate system is viewed, the 3D dot will not be seen as the center of the printed material when this is also a three dimensional one. This does not happen for figure 6, because the text is only a two dimensional object.

5 Nodes

The syntax is

```
\pstThreeDNode(x,y,z) {<node name>}
```

This node is internally transformed into a two dimensional node, so it cannot be used as a replacement for the parameters (x, y, z) of the 3D dot which is possible with the macros from `pst-plot`. If `A` and `B` are two nodes, then `\psline{A}{B}` draws a line from `A` to `B`. Doing the same with `pst-3dplot` is not yet implemented. On the other hand, it is not a problem to define two 3D nodes `C` and `D` and then draw a two dimensional line from `C` to `D`.

6 Dots

The syntax for a dot is

```
\pstThreeDDot [<options>] (x,y,z)
```

Dots can be drawn with dashed lines for the three coordinates, when the option `drawCoor` is set to `true` (figure 7).

```

1 \begin{pspicture}(-2,-2)(2,2)
2 \psset{xMin=-2,xMax=2,yMin=-2,% 
3   yMax=2,zMin=-1,zMax=2,Beta=25}
4 \pstThreeDCoor

```

```

5 \psset{dotstyle=*,dotscale=2,% 
6   linecolor=red,% 
7   drawCoor=true}
8 \pstThreeDDot(-1,1,1)
9 \pstThreeDDot(1.5,-1,-1)
10 \psgrid
11 \end{pspicture}

```

In the figure 8 the coordinates of the dots are (a, a, a) where a is $-3, -2, -1, 0, 1, 2, 3$.

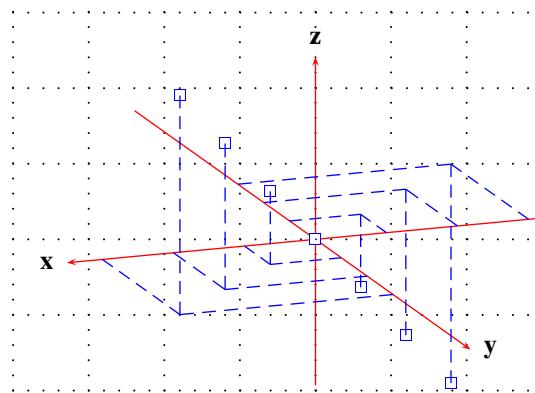


Figure 8: Another demonstration for drawing dots

```

1 \begin{pspicture}(-4,-2)(3,3.25)
2 \psgrid
3 \psset{xMin=-3.5,xMax=3.5,yMin=-7,yMax=6,zMin
4   =-2,zMax=2.5,% 
5   Alpha=20,Beta=15}
6 \pstThreeDCoor
7 \psset{dotstyle=square,dotsize=5pt,% 
8   linecolor=blue,drawCoor=true}
9 \multido{\n=-3+1}{7}{%
10   \pstThreeDDot(\n,\n,\n)% 
11 }
11 \end{pspicture}

```

7 Lines

The syntax for a three dimensional line is

```
\pstThreeDLine [<options>]%
  (x1,y1,z1) (x2,y2,z2)
```

All options for lines from `pst-plot` are possible, there are no special ones for a 3D line. The only difference in drawing a line or a vector is that the first one has an arrow of type `-` and the second type `->` (figure 9).

```

1 \psset{xMin=-2,xMax=2,yMin=-2,yMax=2,% 
2   zMin=-2,zMax=2}
3 \begin{pspicture}(-2,-2.25)(2,2.25)
4 \pstThreeDCoor
5 \psset{dotstyle=*,linecolor=red,% 
6   drawCoor=true}
7 \pstThreeDDot(-1,1,0.5)
8 \pstThreeDDot(1.5,-1,-1)
9 \pstThreeDLine[% 
10   linewidth=3pt,% 
11   linecolor=blue,
12   arrows=->% 
13 ] (-1,1,0.5) (1.5,-1,-1)

```

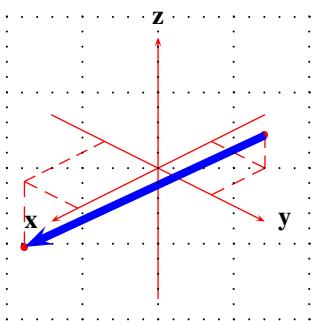


Figure 9: Drawing a 3D vector

```
14      \psgrid
15  \end{pspicture}
```

8 Triangle

A triangle is given by its three points:

```
\pstThreeDTriangle[<options>] (P1) (P2) (P3)
```

When the option `fillstyle` is set to value other than none, the triangle is filled with the active color or with the one which is set with the option `fillcolor` (figure 10).

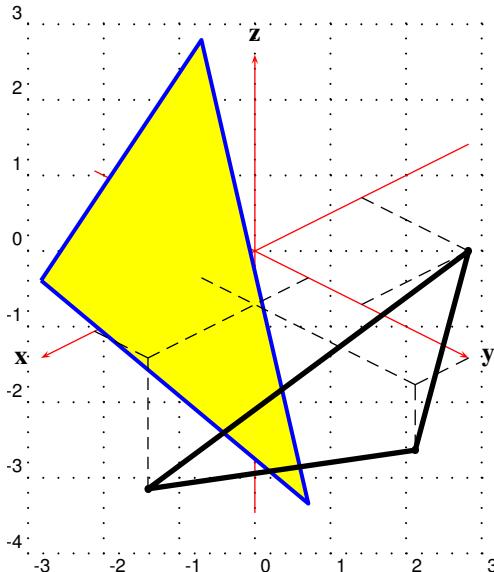


Figure 10: Triangles with fill option

```
1  \begin{pspicture} (-3,-4) (4,3.25)
2    \psgrid
3    \pstThreeDCoor[xMin=-4,xMax=5,yMin=-3,zMin=-4,
                  zMax=3]
4    \pstThreeDTriangle[%
5      fillcolor=yellow,fillstyle=solid,%
6      linecolor=blue,%
7      linewidth=1.5pt](5,1,2)(3,4,-1)(-1,-2,2)
8    \pstThreeDTriangle[%
9      drawCoor=true,linecolor=black,%
```

```
10      linewidth=2pt](3,1,-2)(1,4,-1)(-3,2,0)
11  \end{pspicture}
```

For triangles especially, the option `linejoin` is important. Its value is passed to the PostScript command `setlinejoin`. The default value is 1, which gives rounded edges (figure 11).

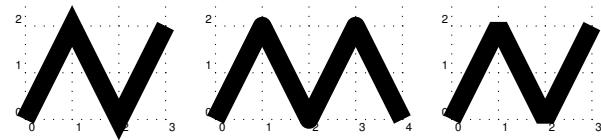


Figure 11: Meaning of the PostScript command `setlinejoin=0|1|2`

9 Squares

The syntax for a 3D square is:

```
\pstThreeDSquare%
[<options>]
(<vector o>)%
(<vector u>) (<vector v>)
```

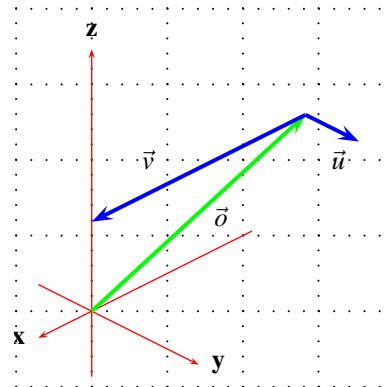


Figure 12: Drawing a square with three vectors

Squares are nothing more than a polygon with the starting point P_o given with the origin vector \vec{o} and the two direction vectors \vec{u} and \vec{v} , which build the sides of the square as shown in figure 12. With the `fillstyle` option the square can be filled with the in `pst-plot` defined styles, for example `solid` like in figure 13. All the options of `pstricks` are allowed for this macro.

```
1  \begin{pspicture} (-3,-2) (4,4)
2    \psgrid
3    \pstThreeDCoor[xMin=-3,xMax=3,yMin=-1,yMax=4,
                  zMin=-1,zMax=4]
4    \pstThreeDSquare[%
5      fillcolor=blue,%
6      fillstyle=solid,%
7      drawCoor=true,dotstyle=*](-2,2,3)(4,0,0)
8      (0,1,0)
9  \end{pspicture}
```

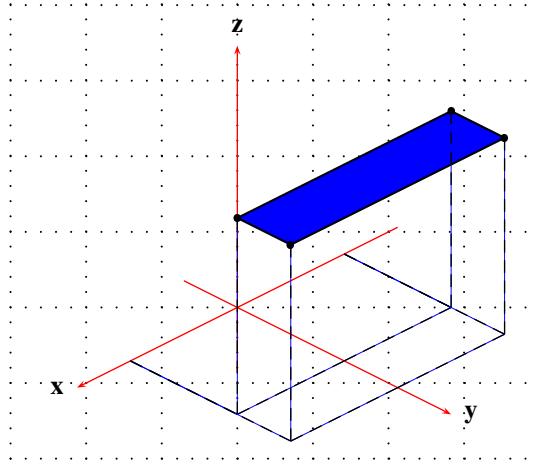


Figure 13: Drawing a filled square with the vectors from figure 12

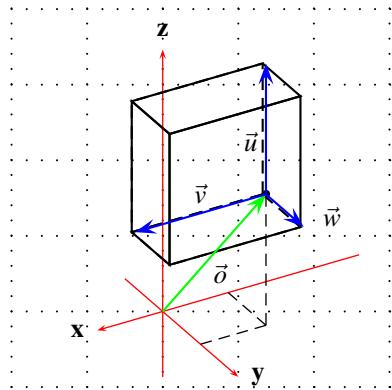


Figure 14: Drawing a box with three vectors

10 Boxes

A box is a special case of a square and has the syntax

```
\pstThreeDBox%
[<options>]
(<vector o>%
 (<vector u>) (<vector v>) (<vector w>)
```

All options from `pstricks` are possible here. The other parameters are the origin vector \vec{o} and the three direction vectors \vec{u} , \vec{v} and \vec{w} . The figure 14 shows a box together with these four vectors. In this example the three direction vectors are perpendicular to each other.

```
1 \begin{pspicture}(-2,-1)(3,4.25)
2   \psgrid
3   \setkeys{psset}{Alpha=30,Beta=30}
4   \pstThreeDCoor[xMin=-3,xMax=1,yMin=-1,yMax=2,
5     zMin=-1,zMax=4]
6   \pstThreeDPut(-1,1,2){\pstThreeDBox(0,0,2)
7     (2,0,0)(0,1,0)}
8   \pstThreeDDot[drawCoor=true](-1,1,2)
9   \setkeys{psset}{arrows=>,arrowsize=0.2}
10  \uput[0](0.5,0.5){$\vec{o}$}
```

```
9   \uput[0](0.9,2.25){$\vec{u}$}
10  \uput[90](0.5,1.25){$\vec{v}$}
11  \uput[45](2,1.){$\vec{w}$}
12  \pstThreeDLine[linecolor=green](0,0,0)(-1,1,2)
13  \pstThreeDLine[linecolor=blue](-1,1,2)(-1,1,4)
14  \pstThreeDLine[linecolor=blue](-1,1,2)(1,1,2)
15  \pstThreeDLine[linecolor=blue](-1,1,2)(-1,2,2)
16 \end{pspicture}
```

11 Ellipses and circles

The equation for a two dimensional ellipse (figure 15) is:

$$e : \frac{(x - x_M)^2}{a^2} + \frac{(y - y_M)^2}{b^2} = 1 \quad (4)$$

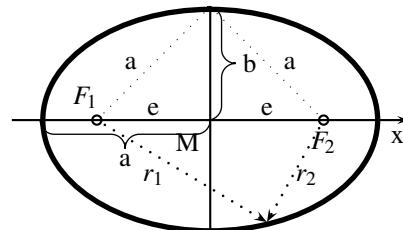


Figure 15: Definition of an ellipse

$(x_m; y_m)$ is the center, a and b the eccentricity. For $a = b = 1$ in equation 4 we get the “one” for the circle, which is nothing more than a special case of an ellipse. The equation written in parametric form is

$$\begin{aligned} x &= a \cdot \cos \alpha \\ y &= b \cdot \sin \alpha \end{aligned} \quad (5)$$

or the same with vectors to get an ellipse in a 3D system:

$$\begin{aligned} e : \vec{x} &= \vec{c} + \cos \alpha \cdot \vec{u} + \sin \alpha \cdot \vec{v} \\ 0 \leq \alpha &\leq 360 \end{aligned} \quad (6)$$

where \vec{c} is the center, \vec{u} and \vec{v} the direction vectors which must be perpendicular to each other.

11.1 Options

In addition to all possible options from the package `pst-plot`, we have two special ones for the drawing of an arc (with predefined values for a full ellipse or circle):

```
beginAngle=0
endAngle=360
```

Using the `parametricplotThreeD` macro (described in section 13.2, ellipses and circles are drawn with a default setting of 50 points for the ellipse or circle).

11.2 Ellipse

In a 3D coordinate system, it is very difficult to see the difference between an ellipse and a circle. Depending on the point of view an ellipse may be seen as a circle and vice versa (figure 16). The syntax of the ellipse macro is:

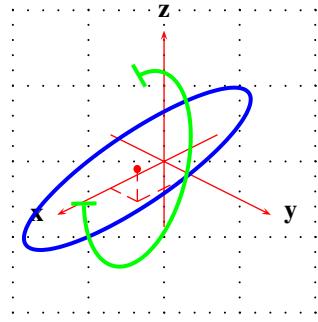


Figure 16: Drawing ellipses

```
\pstThreeDEllipse%
  [<options>]%
  (cx,cy,cz)%
  (ux,uy,uz) (vx,vy,vz)
```

where c is for center and u and v for the two direction vectors (eq. 6).

```
1 \psset{xMin=-1,xMax=2,yMin=-1,yMax=2,zMin=-1,zMax
         =2}
2 \begin{pspicture}(-2,-2)(2,2)
3   \psgrid
4   \pstThreeDCoor
5   \pstThreeDDot[%]
6     linecolor=red,%
7     drawCoor=true](1,0.5,0.5) % the center
8   \pstThreeDEllipse[%]
9     linecolor=blue, linewidth=1.5pt]%
10    (1,0.5,0.5) (-0.5,1,0.5) (1,-0.5,-1)
11  % settings for an arc
12  \pstThreeDEllipse[%]
13    beginAngle=0,endAngle=270,%
14    linecolor=green]%
15    (1,0.5,0.5) (-0.5,0.5,0.5) (0.5,0.5,-1)
16 \end{pspicture}
```

11.3 Circle

The circle is a special case of an ellipse (eq. 6) with the vectors \vec{u} and \vec{v} which are perpendicular to each other: $|\vec{u}| = |\vec{v}| = r$. with $\vec{u} \cdot \vec{v} = \vec{0}$

The macro `\pstThreeDCircle` is nothing more than a synonym for `\pstThreeDEllipse`. In the following example the circle is drawn with only 20 plot-points and the option `showpoints=true`.

```
1 \begin{pspicture}(-2,-1)(2,2)
2   \psgrid
3   \pstThreeDCoor[%]
4     xMin=-1,xMax=2,yMin=-1,yMax=2,zMin=-1,zMax
         =2,%
5     linecolor=black]
6   \pstThreeDCircle[%]
7     linecolor=red, linewidth=2pt,%
8     plotpoints=20, showpoints=true]%
9     (1.6,+0.6,1.7) (0.8,0.4,0.8) (0.8,-0.8,-0.4)
10  \pstThreeDDot[drawCoor=true, linecolor=blue
                  ](1.6,+0.6,1.7)
11 \end{pspicture}
```

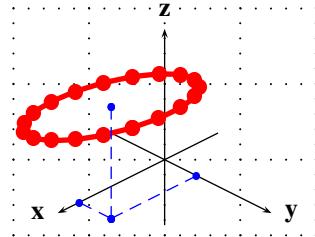


Figure 17: Drawing a circle with the option `showpoints`

12 Spheres

Internally, `pst-3dplot` uses the macro from the `pst-vue3d` package¹ to draw spheres, and places it with the `\rput` macro at the right place. The syntax for this macro is

```
\pstThreeDSphere[<options>] (x,y,z) {Radius}
```

(x,y,z) is the center of the sphere. For all the other possible options or the possibility to draw demi-spheres, refer to the documentation.[3]

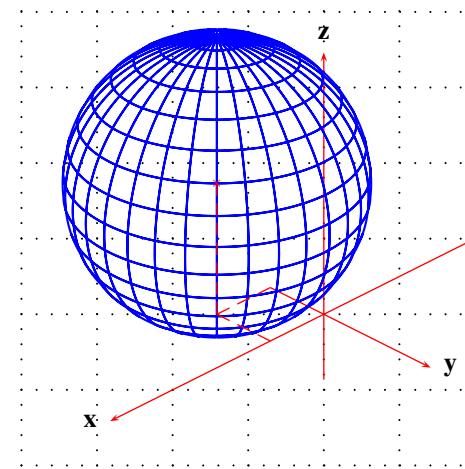


Figure 18: Drawing a sphere with package `pst-vue3d`

```
1 \begin{pspicture}(-4,-2)(2,4)
2   \psgrid
3   \pstThreeDCoor[xMin=-3,xMax=4,yMin=-1,yMax=2,
                 zMin=-1,zMax=4]
4   \pstThreeDSphere[linecolor=blue](1,-1,2){2}
5   \pstThreeDDot[dotstyle=x, linecolor=red, drawCoor
                  =true](1,-1,2)
6 \end{pspicture}
```

¹ CTAN:graphics/pstricks/contrib/pst-vue3d, and from Manuel Luque's homepage[3]. The documentation is in French, but it is mostly self-explanatory.

13 Mathematical functions

There exist two macros for plotting mathematical functions $f(x,y)$, which work similarly to the one from `pst-plot`.

13.1 Function $f(x,y)$

The macro for plotting functions does not have the same syntax as the one from `pst-plot[5]`, but it is used in the same way:

```
\psplotThreeD[<options>]%
  (xMin, xMax) (yMin, yMax) %
  {<the function>}
```

The function has to be written in PostScript code and the only valid variable names are `x` and `y`. For example, `{x dup mul y dup mul add sqrt}` represents the math expression $\sqrt{x^2 + y^2}$. The macro `\psplotThreeD` has the same plotstyle options as `\psplot`, except the `plotpoints`-option which is split into one for `x` and one for `y` (table 1).

Table 1: Options for the plot macros

Option name	value
<code>plotstyle</code>	<code>dots</code> <code>line</code> <code>polygon</code> <code>curve</code> <code>ecurve</code> <code>ccurve</code> none (default)
<code>showpoints</code>	default is false
<code>xPlotpoints</code>	default is 25
<code>yPlotpoints</code>	default is 25
<code>hiddenLine</code>	default is false

Equation 7 is plotted with the following parameters and seen in figure 19.

$$z = 10 \left(x^3 + xy^4 - \frac{x}{5} \right) e^{-(x^2+y^2)} + e^{-((x-1.225)^2+y^2)} \quad (7)$$

```

1  \begin{pspicture}(-6,-4)(6,5)
2    \psgrid
3    \psset{Alpha=45,Beta=15}
4    \psplotThreeD[%
5      plotstyle=line,%
6      yPlotpoints=40,xPlotpoints=30,%
7      linewidth=1pt]{-4,4}{-4,4}{%
8        x 3 exp x y 4 exp mul add x 5 div sub
9        10 mul
10       2.729 x dup mul y dup mul add neg exp
11       mul
12       2.729 x 1.225 sub dup mul y dup mul add
13       neg exp add}
11     \pstThreeDCoor[xmin=-1,xMax=5,yMin=-1,yMax=5,
12     zMin=-1,zMax=5]
12   \end{pspicture}
```

The function is calculated within two loops:

```
for (float y=yMin; y<yMax; y+=dy)
  for (float x=xMin; x<xMax; x+=dx)
    z=f(x,y);
```

Because of the inner loop it is only possible to get a closed curve in `x` direction. Therefore fewer `yPlotpoints` are not a real problem, but too few `xPlotpoints` results in a bad drawing of the mathematical function, especially for the `plotstyle` option `line`.

Drawing three dimensional mathematical functions with curves which are transparent makes it difficult to see if a point is before or behind another one. `\psplotThreeD` has an option `hiddenLine` for a primitive hidden line mode, which only works well when the `y`-interval is defined such that $y_2 > y_1$. Then, every new curve is plotted over the previous one and filled with the color white. Figure 20 is the same as figure 19, only with the option `hiddenLine=true`.

13.2 Parametric plots

Parametric plots are possible for drawing curves or areas. The syntax for this plot macro is:

```
\parametricplotThreeD[<options>]%
  (t1,t2) (u1,u2) %
  {<three parametric functions x y z}
```

The only possible variables are `t` and `u` with `t1,t2` and `u1,u2` as the range for the parameters. The order for the functions is not important and `u` may be optional when having only a three dimensional curve and not an area.

$$\begin{aligned} x &= f(t,u) \\ y &= g(t,u) \\ z &= h(t,u) \end{aligned} \quad (8)$$

To draw a spiral we have the parametric functions:

$$\begin{aligned} x &= r \cos t \\ y &= r \sin t \\ z &= t/600 \end{aligned} \quad (9)$$

In the example, the `t` value is divided by 600 for the `z` coordinate, because we have the values for `t` in degrees, here with a range of $0^\circ \dots 2160^\circ$. Drawing a curve in a three dimensional coordinate system does only require one parameter, which is by default `t`. In this case we do not need all parameters, so that we can write

```
\parametricplotThreeD[<options>]%
  (t1,t2) %
  {<three parametric functions x y z}
```

which is the same as $(0,0)$ for the parameter `u`. Figure 21 shows a three dimensional curve.

```

1  \begin{pspicture}(-3,-2)(3,5)
2    \psgrid
3    \parametricplotThreeD[%
4      xPlotpoints=200,%
5      linecolor=blue,%
```

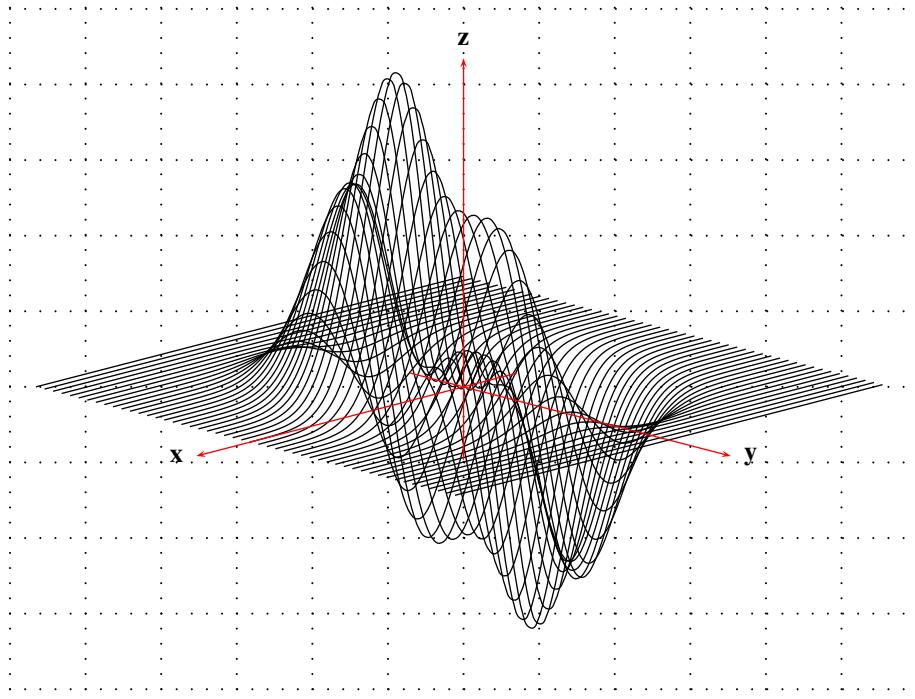


Figure 19: Plot of equation 7

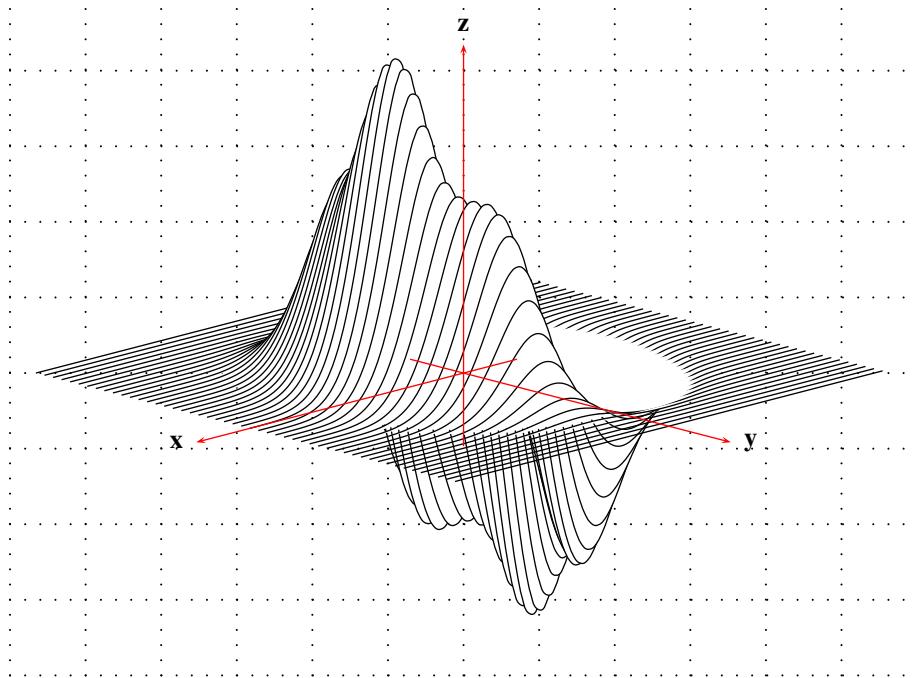


Figure 20: Plot of equation 7 with the `hiddenLine=true` option

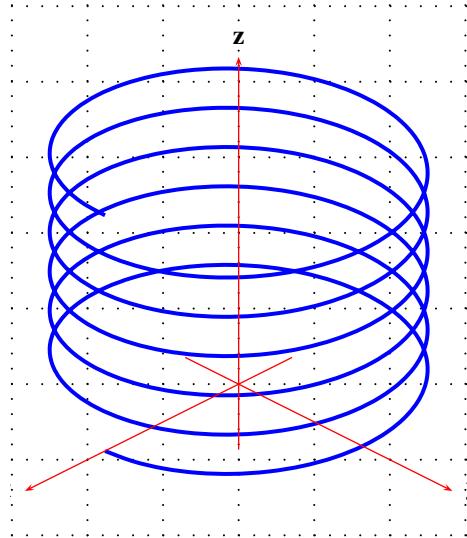


Figure 21: Drawing a 3D curve

```

6      linewidth=1.5pt,
7      plotstyle=curve] (0,2160) {%
8          2.5 t cos mul
9          2.5 t sin mul
10         t 600 div%
11     }
12 \pstThreeDcoor[xMin=-1,xMax=4,yMin=-1,yMax=4,
13                  zMin=-1,zMax=5]
14 \end{pspicture}

```

Instead of using the `\pstThreeDSphere` macro (see section 12) it is also possible to use parametric functions for a sphere. The macro plots continuous lines only for the t parameter, so a sphere plotted with the longitudes needs the parametric equations as

$$\begin{aligned} x &= \cos t \cdot \sin u \\ y &= \cos t \cdot \cos u \\ z &= \sin t \end{aligned} \quad (10)$$

The same is possible for a sphere drawn with the latitudes:

$$\begin{aligned} x &= \cos u \cdot \sin t \\ y &= \cos u \cdot \cos t \\ z &= \sin u \end{aligned} \quad (11)$$

and lastly, we can have both of these parametric functions together in one `pspicture` environment (figure 22).

```

1 \begin{pspicture}(-1,-1)(1,1)
2 \psgrid
3 \parametricplotThreeD[%
4   plotstyle=curve,yPlotpoints=40](0,360)(0,360){%
5     t cos u sin mul
6     t cos u cos mul
7     t sin
8   }
9 \parametricplotThreeD[%
10  plotstyle=curve,yPlotpoints=40](0,360)(0,360){%
11    u cos t sin mul
12    u cos t cos mul
13    u sin
14  }
15 \end{pspicture}

```

```

14   }
15 \end{pspicture}

```

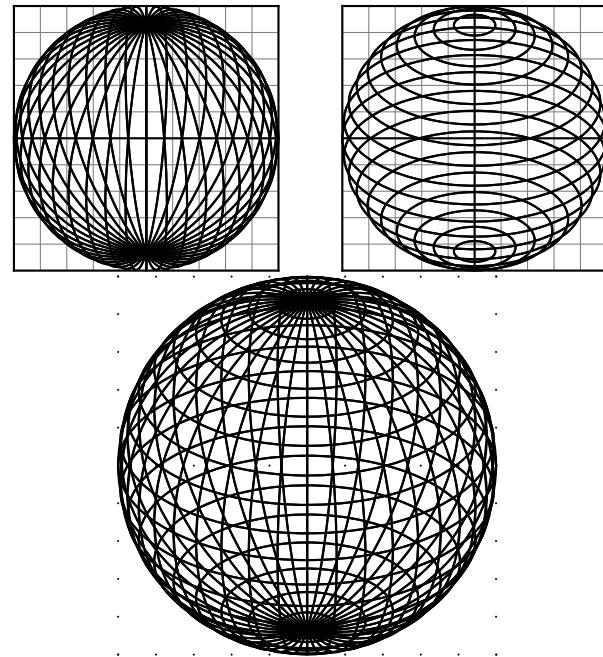


Figure 22: Different views of the same parametric functions

14 Plotting data files

We have the same conventions for data files which hold 3D coordinates as for 2D. For example:

```

0.0000  1.0000  0.0000
-0.4207  0.9972  0.0191
...
0.0000, 1.0000, 0.0000
-0.4207, 0.9972, 0.0191
...
(0.0000,1.0000,0.0000)
(-0.4207,0.9972,0.0191)
...
{0.0000,1.0000,0.0000}
{-0.4207,0.9972,0.0191}
...

```

There are the same three plot functions:

```

\fileplotThreeD[<options>]{<datafile>}
\dataplotThreeD[<options>]{<data object>}
\listplotThreeD[<options>]{<data object>}

```

The data file used in the following examples has 446 entries like

```

6.26093349..., 2.55876582..., 8.131984...

```

Using the `listplotThreeD` macro with many data entries may take considerable time on slow machines. The possible options for the lines are the same as earlier, given in table 1.

14.1 `\fileplotThreeD`

The syntax is straightforward:

```
\fileplotThreeD[<options>]{<datafile>}
```

If the data file is not in the same directory as the document, use the file name with the full path. Figure 23 shows a file plot with the option `linestyle=line`.

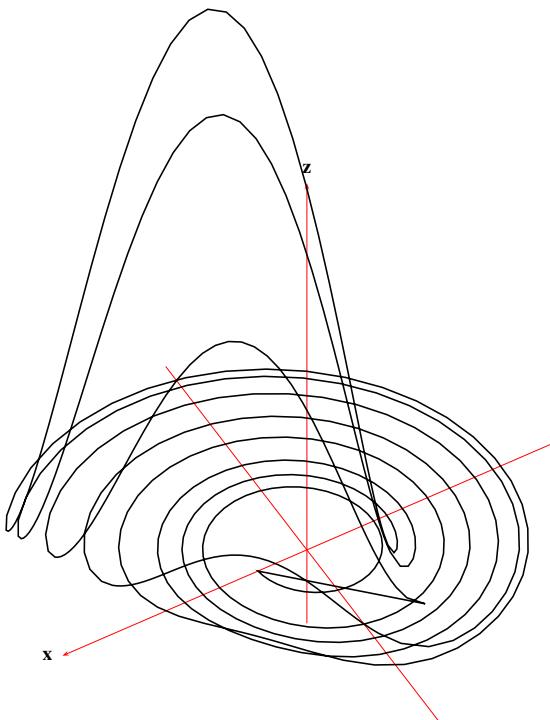


Figure 23: Demonstration of `\fileplotThreeD` with Alpha=30 and Beta=15

```

1  \begin{pspicture}(-7.5,-3)(6,10)
2    \psset{xunit=0.5cm,yunit=0.75cm,%
3           Alpha=30,Beta=30} % the global parameters
4    \pstThreeDCoor[% 
5      xMin=-10,xMax=10,% 
6      yMin=-10,yMax=10,% 
7      zMin=-2,zMax=10]
8    \fileplotThreeD[plotstyle=polygon]{data3D.
9      Roessler}
9  \end{pspicture}

```

14.2 `\dataplotThreeD`

The syntax is:

```
\dataplotThreeD[<options>]{<data object>}
```

In contrast to `\fileplotThreeD`, the second macro `\dataplotThreeD` reads the data entries from another

macro. Using `\readdata`, external data can be read from a file and saved in a macro, to be passed to `\dataThreeD[1]`.

```
\readdata{<data object>} {<datafile>}
```

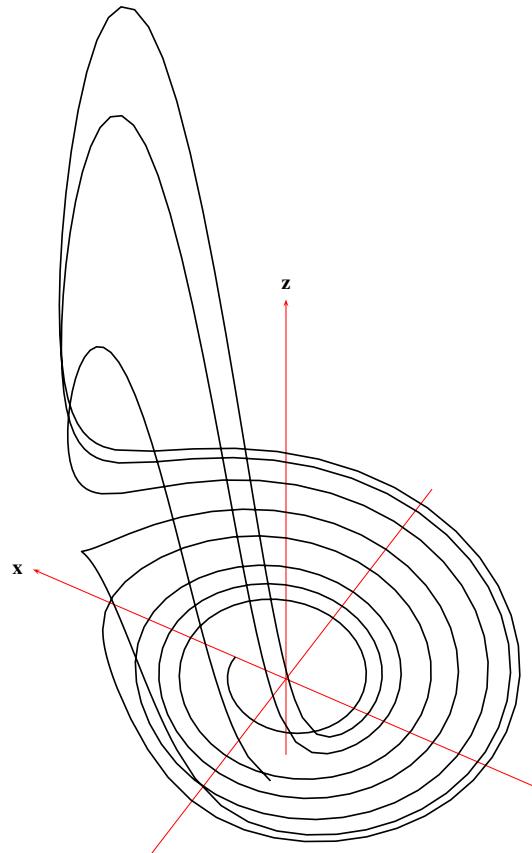


Figure 24: Demonstration of `\dataplotThreeD` with Alpha=-30 and Beta=30

```

1  \readdata{\dataThreeD}{data3D.Roessler} [...]
2  \begin{pspicture}(-6,-2.25)(6,11)
3    \psset{xunit=0.5cm,yunit=0.75cm,%
4           Alpha=-30}
5    \pstThreeDCoor[% 
6      xMin=-10,xMax=10,% 
7      yMin=-10,yMax=10,% 
8      zMin=-2,zMax=10]
9    \dataplotThreeD[plotstyle=line]{\dataThreeD}
10 \end{pspicture}

```

14.3 `\listplotThreeD`

The syntax is:

```
\listplotThreeD[<options>]{<data object>}
```

There is no essential difference between the macros `\listplotThreeD` and `\dataplotThreeD`. With `\listplotThreeD`, one can pass additional PostScript code, which is appended to the data object. For example:

```

1 \dataread{\data}{data3D.Roessler}
2 \newcommand{\dataThreeDDraft}{%
3   \data\space
4   gsave           % save graphic state
5   /Helvetica findfont 40 scalefont setfont
6   45 rotate       % rotate 45 degrees
7   0.9 setgray     % 1 ist white
8   -60 30 moveto (DRAFT) show
9   grestore
10 }

```

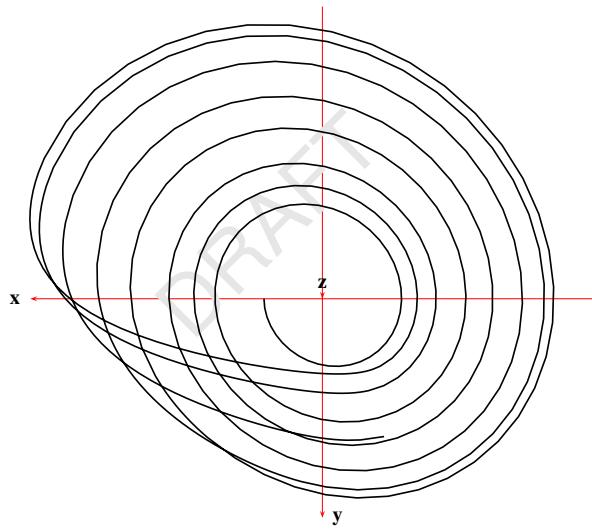


Figure 25: Demonstration of `\listplotThreeD` with a view from above ($\Alpha=0$ and $\Beta=90$) and some additional PostScript code

Figure 25 shows what happens with this additional PostScript code. Another example can be found in [5], where `ScalePoints` is redefined. For `pst-3dplot`, the equivalent macro is named `ScalePointsThreeD`.

```

1 \begin{pspicture}(-5,-4)(5,4.5)
2   \psset{xunit=0.5cm,yunit=0.5cm,%
3          Alpha=0,Beta=90}
4   \pstThreeDCoor[%
5     xMin=-10,xMax=10,%]
6     yMin=-10,yMax=7.5,%]
7     zMin=-2,zMax=10]
8   \listplotThreeD[plotstyle=line]{%
9     dataThreeDDraft}
9 \end{pspicture}

```

15 PDF output

`pst-3dplot` is based on the popular `pstricks` package and writes pure PostScript code[2], so it is not possible to run `TEX` files with `pdflatEX` when there are `pstricks` macros in the document. If you need PDF output, there are the following possibilities:

- the package `pdftricks.sty` [6]
- the free (for Linux only) program `VTEX/Lnx` (<http://www.micropress-inc.com/linux/>)
- the `ps2pdf` (`dvi→ps→pdf`) or `dvipdfm` utilities
- the `ps4pdf` package [4].

If you need package `graphicx.sty`, load it before any `pstricks` package. You do not need to load `pstricks.sty`, as this will be done by `pst-3dplot`.

References

- [1] Laura E. Jackson and Herbert Voß. Die Plot-Funktionen von `pst-plot`. *Die T_EXnische Komödie*, 2/02:27–34, June 2002.
- [2] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. IWT, Vaterstetten, 1989.
- [3] Manuel Luque. *Vue en 3D*. <http://members.aol.com/Mluque5130/vue3d16112002.zip>, 2002.
- [4] Rolf Niepraschk. *ps4pdf*. CTAN:/macros/latex/contrib/ps4pdf/, 2003.
- [5] Herbert Voß. Die mathematischen Funktionen von PostScript. *Die T_EXnische Komödie*, 1/02:40–47, March 2002.
- [6] Herbert Voß. *PSTricks Support for pdf*. <http://www.educat.hu-berlin.de/~vooss/lyx/pdf/pdftricks.phtml>, 2002.
- [7] Timothy van Zandt. *PSTricks - PostScript macros for Generic T_EX*. <http://www.tug.org/application/PSTricks>, 1993.

◊ Herbert Voß
Wasgenstr. 21
14129 Berlin GERMANY
vooss@perce.de
<http://www.perce.de>

Axis alignment in XY-pic diagrams

Alexander R. Perlis

Abstract

By default, XY-pic aligns diagrams according to the *center* of each object. For many types of diagrams, such *center alignment* is the preferred choice; however, *axis alignment* is sometimes better. For example, compare $A \rightarrow A^2$ (center-aligned) with $A \rightarrow \overline{A^2}$ (axis-aligned); the arrow stayed in the same place, but the A moved up a little, and the A^2 moved up a lot. Note that the simple T_EX code $\$A \rightarrow A^2 \$$ uses axis alignment: $A \rightarrow \overline{A^2}$.

This article studies attempts to instruct XY-pic to use axis alignment and presents a concise solution to the problem. Enhancements to XY-pic are proposed.

1 Preliminaries

The version of XY-pic used here is 3.7 (16Feb1999). The *XY-pic User's Guide* will be cited as [XY GUIDE], and the *XY-pic Reference Manual* as [XY MANUAL]. These documents are part of the XY-pic distribution available on CTAN.

This article's abstract already defined *center alignment* and exhibited the subtle differences between that and *axis alignment*, but the latter was left undefined. For now, it means "the alignment used by the simplest of T_EX code". Matters will make more sense after section 5, where the alignment practices of T_EX and XY-pic are explained in detail.

For most of this article, we will study attempts at axis alignment using the `\xymatrix` feature of XY-pic. (It is introduced nicely in [XY GUIDE].) The *one-line solution* to our alignment problem appears in section 7. Solutions for the `\xygraph` feature and for XY-pic kernel code appear there as well.

2 The problem

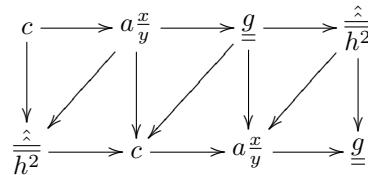
We are (happily) compelled to use XY-pic because it produces fantastic diagrams we cannot otherwise obtain, yet we recognize that many of our center-aligned diagrams ought to be axis-aligned. We wish to instruct XY-pic to use the preferred alignment.

To study the matter, we need a toy example that exhibits a wide gulf between the two types of alignment, yet fits in this article's columns.

```
\def\toyone{c} \def\toytwo{\frac{x}{y}}
\def\toythree{\underline{\underline{g}}}
\def\toyfour{\hat{h}\hat{\overline{h^2}}}
\def\toyexample{%
```

```
\toyone \ar[d] \ar[r]
& \toytwo \ar[d] \ar[r] \ar[dl]
& \toythree \ar[d] \ar[r] \ar[dl]
& \toyfour \ar[d] \ar[dl] \\
\toyfour \ar[r]
& \toyone \ar[r]
& \toytwo \ar[r]
& \toythree \\
}
```

The result of `\xymatrix{\toyexample}` is



What a monstrosity! To shirk responsibility, let's take the viewpoint that a famous mathematician has hired us to typeset this bewildering diagram as part of her new book. The notation is beyond our control.

A quick peek (go ahead!) at the end of section 7 shows what we're after. To get a sense of the difference, compare the top rows of the two diagrams without the arrows:

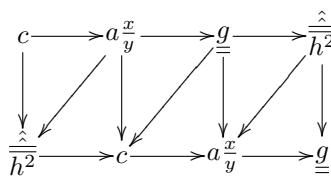
$$ca_x^y \underline{\underline{g}} \hat{\overline{h^2}} \text{ versus } ca_x^y \underline{\underline{g}} \hat{\overline{h^2}}.$$

The reader interested merely in the solution to our problem should skip directly to section 7, or to section 5 for some background on that solution.

The material below in sections 3 and 4 has nothing to do with the ultimate solution, and is included mainly for the XY-pic enthusiast interested in why `\xymatrix@1` and variations on that theme do not solve our problem.

3 Using `\xymatrix@1`

The first idea is to try `\xymatrix@1` in place of `\xymatrix`. This gives



The top row is

$$ca_x^y \underline{\underline{g}} \hat{\overline{h^2}}.$$

That's neither center-aligned nor axis-aligned! To be fair, [XY GUIDE, §1.4, p.3] only encourages use of `\xymatrix@1` with one-line diagrams. Leaving our toy example aside for the rest of this section, let's

experiment a bit with one-line diagrams:¹

```
$\xymatrix{A \ar[r] & A'}$       $A \rightarrow A'$ 
```

```
$\xymatrix@1{A \ar[r] & A'}$       $A \rightarrow A'$ 
```

Hoorah! The latter is axis-aligned! But if we replace A' with A^2 , we get:

```
$\xymatrix{A \ar[r] & A^2}$       $A \rightarrow A^2$ 
```

```
$\xymatrix@1{A \ar[r] & A^2}$       $A \rightarrow A^2$ 
```

Unhoorah. This time the result is neither center-aligned nor axis-aligned: the placement of A^2 is too low. The mistake is easier to spot by enlarging and putting all the diagrams on a rule:

$$\begin{array}{ccc} A \rightarrow A' & A \rightarrow A' & A \rightarrow A' \\ A \rightarrow A^2 & A \rightarrow A^2 & A \rightarrow A^2 \end{array}$$

Left: `\xymatrix`. Middle: `\xymatrix@1`. For comparison, `\$A \rightarrow A'` and `\$A \rightarrow A^2` are included on the right—being the simplest of TeX code, they are axis-aligned by definition!

In one case of a one-line diagram, `\xymatrix@1` succeeds, yet in another, it fails. Why did that happen? The difference between `\xymatrix` and `\xymatrix@1` is that the latter inserts a zero-width left parenthesis at the start of every entry, and doing so affects the vertical spacing.²

$$\begin{array}{ccc} A \rightarrow A' & (A \rightarrow A' & A \rightarrow A' \\ A \rightarrow A^2 & (A \rightarrow (A^2 & A \rightarrow A^2 \end{array}$$

An entry's center is determined by its bounding box, which in turn is determined by the parts of the entry that stick out the most. Both A and A' are dwarfed by the parenthesis, but A^2 is not: the superscript sticks out more than the parenthesis. Consequently, the center of $(A^2$ is slightly higher than that of $(A$ or $(A'$; thus, to align entries by their center, XY-pic must lower $(A^2$ slightly. (Why doesn't it instead raise everything else? We'll answer this question in section 5.)

In summary, `\xymatrix@1` gives an axis-aligned result only when the diagram's entries fit inside regular parentheses. Otherwise, the result likely will be neither center-aligned nor axis-aligned.

¹ The diagrams in `...$` were made smaller by setting up `\xymatrix` with `@-1.25pc@M=1pt`. The sample code does not reflect this.

² There is another difference between `\xymatrix` and `\xymatrix@1`, not documented in [XY MANUAL]: `@1` implies `@M=1pt`. This explains why, in comparing the diagram at the start of section 3 to the one at the end of section 2, the arrows are closer to the entries.

4 Mimicking `\xymatrix@1`

Ah ha! With large entries, the zero-width left parenthesis inserted by `\xymatrix@1` is not tall enough, so let's use a taller one. Since

```
\xymatrix@1{\toyexample}
```

is equivalent² to

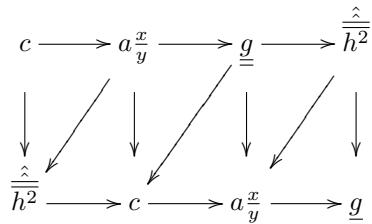
```
\everyentry={\vphantom{}}
\xymatrix{\toyexample}
```

we might first try

```
\everyentry={\vphantom{\bigl(}}
\xymatrix{\toyexample}
```

but discover this isn't enough, and after running out of named sizes, we might try letting TeX make the precise calculation:

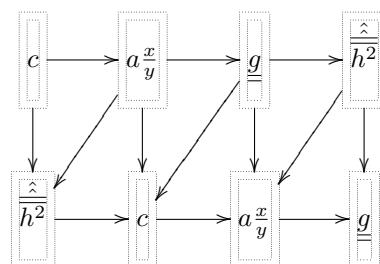
```
\everyentry={\vphantom{\left(\toyone\toytwo
\toythe\toytfour\right.)}}
\xymatrix{\toyexample}
```



Hey, this is axis-aligned! Unfortunately, many of the arrows now appear to be afraid of the entries. To understand what went wrong, take a look at the size of the delimiter we inserted around each entry:

```
\left(\toyone\toytwo
\toythe\toytfour\right). \left(\begin{matrix} x \\ ca \\ y \end{matrix}\right)
```

Each entry's vertical size is determined by the delimiter, and then, as usual, `\xymatrix` adds an additional margin:



Evidently, the promising approach of inserting zero-width material to affect vertical alignment, which is used by `\xymatrix@1`, is *fundamentally flawed*: important height information gets lost!

It's time to step back and study the alignment algorithms in TeX and XY-pic.

5 How TeX and XY-pic align objects

Earlier we asked: in going from $A \rightarrow A'$ to $A \rightarrow A'$, why does the arrow stay put and the A and A' move up, instead of, say, the A staying put, the arrow moving down, and the A' moving (slightly) up?

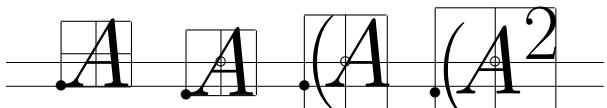
Inside math mode, TeX maintains two reference lines for alignment purposes: the *baseline* and the *axis*. These lines are not part of the characters being typeset; rather, they depend only on the current font and thus should be thought of as being part of the underlying canvas. As for the characters, each one has a bounding box and a reference point (obtained from the font's TFM file).

In horizontal mode and in math mode, TeX positions each character so that the character's reference point lands on the canvas's baseline. The axis comes into play when TeX builds a fraction: the numerator and denominator are positioned so that the bar of the fraction lands on the canvas's axis. Each delimiter, such as the left parenthesis, is designed to involve both lines: as with all characters, TeX positions the delimiter so that its reference point lands on the canvas's baseline; however, in so doing, due to the shape of the delimiter, the middle of the delimiter lands precisely on the canvas's axis. In other words, after placement, each delimiter has equal height and depth *when measured from the axis*, but not when measured from the baseline.

XY-pic, on the other hand, maintains its own reference point for each object, which starts out in the *center* of the object. When XY-pic hands a finished object to TeX for placement, it does so in such a way that the object's XY-pic reference point lands on the canvas's axis.³

Let's summarize. When TeX is in charge, the TeX reference point lands on the baseline. When XY-pic is in charge, the XY-pic reference point lands on the axis.

Let's illustrate. On the left, we see how TeX positions an A : the TeX reference point • lands on the baseline.

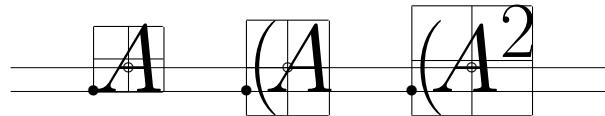


Next we see how XY-pic positions A , $(A$, and $(A^2$: the XY-pic reference point ◦, which defaults to the object's center, lands on the axis. In the case of $(A$, the TeX reference point happens to land on the base-

³ As explained in [XY MANUAL, §2.1, p.6], that's true only when XY-pic was entered inside math mode. Otherwise there is no axis, and so the canvas's baseline is used. This affects *everything*. The net effect is to shift the entire diagram, arrows and all, by a fixed amount.

line, but that's merely a consequence of how delimiters are designed (discussed earlier in this section).

We wish to achieve the following:



The XY-pic reference point should be positioned away from the center in such a way that when it lands on the axis, the TeX reference point will land on the baseline. The crucial measurement is easy to spot: the vertical distance between the XY-pic reference point and the TeX reference point should be the same as the distance between the canvas's baseline and axis. TeX will cough up that distance if you feed it `\fontdimen22\textfont2`.

By positioning the XY-pic reference point appropriately, we achieve the desired alignment without changing the object's bounding box. This is the solution we've been seeking! The code is presented in section 7.

6 Aside: the term *axis-aligned*

By the discussion in the previous section, we conclude that *axis-aligned* means: objects are aligned with their TeX reference point on the baseline, while diagram arrows point to the axis (because that's where the XY-pic reference point is). But the following description does a better job justifying the term. Before being dropped on the canvas, each object is typeset in its own box and thus has its own axis. Getting the object's TeX reference point onto the canvas's baseline is equivalent to getting the object's axis onto the canvas's axis. Thus *axis-aligned* means: each object's axis lies on the canvas's axis, and each arrow points to that common axis. In short, everything is aligned by the axis!

7 The solution

We have seen that XY-pic positions an object so that the XY-pic reference point, which defaults to the object's center, lands on the canvas's axis. To alter the placement, we either move the object's center by changing its size, or move the XY-pic reference point away from the center. In section 4 we disposed of the idea of changing the object's size, because the original size is needed later for drawing arrows. At the end of section 5, we saw that moving the XY-pic reference point appropriately solves our problem.

All said and done, our solution is to put `\entrymodifiers={+!!<\Opt,\fontdimen22\textfont2>}` prior to each `\xymatrix`, or simply once and for all in the document's preamble.

The `+` sets up a margin similar to the default margin used by `\xymatrix` (the difference is discussed briefly in section 9). The first `!` moves the `Xy-pic` reference point from the object's center down to the line containing `TeX`'s reference point, and then `!<0pt,\fontdimen22\textfont2>` moves it up the appropriate distance so that, when it is dropped on the canvas's axis, the `TeX` reference point lands on the canvas's baseline.

Although [XY GUIDE] introduces `Xy-pic` in terms of the `\xymatrix` feature, there are other ways of using `Xy-pic`, notably `\xygraph` or even direct kernel code. With the `\xygraph` feature, axis alignment is obtained by putting

```
!~{*+!!<0pt,\fontdimen22\textfont2>}
```

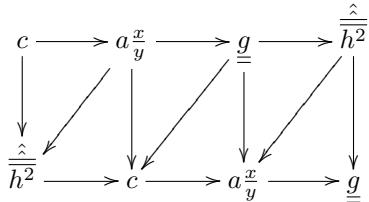
at the start of each graph. With `Xy-pic` kernel code, add the drop modifiers

```
!!<0pt,\fontdimen22\textfont2>
```

to each object that should be axis-aligned. But be careful: the effect is cumulative. Thus, if you drop a `\composite` of objects that should be axis-aligned, either axis-align the composite object, or the individual objects, but not both.

Returning to our toy example from section 2, the axis-aligned result is:

```
\entrymodifiers={+!!<0pt,\fontdimen22\textfont2>}
\xymatrix{\toyexample}
```



8 Caveats

8.1 Size

By default, `Xy-pic` builds objects in `\textstyle`.⁴ If, say, `\scriptstyle` is used instead, then each use of `\textfont2` in our solution should be replaced with `\scriptfont2`. After all, the distance between the baseline and axis depends on the size of the math font.

8.2 Labels

By default, `Xy-pic` places arrow labels halfway between the `Xy-pic` reference points of the source and destination objects. Even though the shenanigans in this article move the reference point away from the object's center, from the viewpoint of the un-

⁴ [XY MANUAL, §4] incorrectly claims the default to be `\displaystyle`. To actually obtain `\displaystyle`, one puts `\objectstyle=\displaystyle`.

derlying canvas, it is the object that moves, not the reference point! On the canvas, the final locations of the reference points remain the same, and thus all labels and arrow destinations remain put. However, objects shift vertically, thus affecting their bounding boxes and the *lengths* of arrows. That in turn affects our perception of whether a label is properly placed. In short, hand-tuned code that does a great job with labels on a center-aligned diagram may not do a great job on an axis-aligned diagram. Moral: first settle on a choice of diagram alignment, then tune the placement of your labels.

9 Proposed enhancements to `Xy-pic`

Both from the public `Xy-pic` list

<http://tug.org/mailman/listinfo/xy-pic>

and from private email exchanges, I gather that the authors of `Xy-pic` welcome the discussion of ideas for improving `Xy-pic`. Perhaps someone familiar with the source code of `Xy-pic` could experiment with implementations of the following ideas.

1. The kernel language might support the drop modifier `!A` to have the same effect as

```
!<0pt,\fontdimen22\textfont2>.
```

(The letter ‘A’ reminds us of “axis” and “alignment”.) Actually, the definition should depend on `\objectstyle`. For example, with

```
\objectstyle=\scriptstyle,
```

`!A` should be shorthand for

```
!<0pt,\fontdimen22\scriptfont2>.
```

2. The `\xymatrix` feature could support `@A` as a setup to have the same effect as setting

```
\entrymodifiers={+!!A}.
```

Actually, the source code indicates that the default value is `\entrymodifiers={\entrybox}`, and the source for `\entrybox` seems to do more than `\entrymodifiers={+}` would do. Is that true? If so, `@A` should probably also use the more complicated behavior. The point is to gain axis alignment without losing something else.

3. The setup `@1` could be redefined to simply mean `@A@M=1pt`. As discussed in section 3, today's `@1` is a buggy construct: the math strut negatively affects vertical spacing and arrows; in particular, today's `@1` even fails to properly align simple one-line diagrams like $A \rightarrow A^2$.
4. The matrix option might support some kind of global `\everyxymatrix={...}`, so that one can easily specify setups like `@A` once and for all in a document's preamble. The alternative is to put

\entrymodifiers={+!!\texttt{A}} in the preamble, but newcomers to XY-pic are likely to master the use of \xymatrix setups prior to tackling kernel-level drop modifiers.

5. Similarly, the graph option might support \everyxygraph=\{\ldots\}. When missing, the usual defaults [XY MANUAL, p.53] would apply.

10 Acknowledgments

The problem of aligning objects and arrows appropriately to achieve beautiful diagrams is hardly new. (Just think of TeX itself, or all the diagram packages available on CTAN.) Within the context of XY-pic, the problem was discussed and solved in 2001 on the XY-pic list by Vadim Radionov, whose solution is essentially identical to the one obtained here.

I thank Ross Moore and Florian Lengyel for commenting on earlier versions of this article, and Michael Abbott for discussing the ideas in section 9.

◊ Alexander R. Perlis
Department of Mathematics
The University of Arizona
Tucson, AZ 85721 USA
aprl@math.arizona.edu

ics teachers would say to describe geometric figures. For instance, the former problem, written as an exercise, could be:

Let ABC be a triangle and \mathcal{I} its inscribed circle. Draw ABC and \mathcal{I} .

In Eukleides, it gives:

```
A B C triangle
I = incircle(A,B,C)
draw(A,B,C) ; draw(I)
```

Which leads to the following graphical result:

```
frame(-0.5,-0.5,6.5,4.5) A B C triangle I =
incircle(A,B,C) draw(A,B,C) ; draw(I)
```

Once the design of the language was done, I wrote **eukleides**,¹ a compiler which translates Eukleides code into *PSTricks* macros. This program can run as a filter. That is, it can take a *LATeX* source containing Eukleides code, and replace this code with *PSTricks* macros, producing a ready-to-*TeX* file.

There's also a graphical interface to the language, with additional interactive features, named **xeukleides**. It was first meant for classroom presentations, but it can also be seen as a tool to compose and tune some Eukleides code for later inclusion in a *LATeX* source.

Both programs are released under the GNU Public License. They were developed on a GNU/Linux system, and were ported to several operating systems: NetBSD, FreeBSD, Mac OS X, MS Windows. Their source code is available from CTAN² or the Eukleides home page³ (which also offers GNU/Linux and Win32 executables).

Around Morley's triangle

As a first introduction to the Eukleides language, we'll study the source code which gives the following figure. It illustrates Morley's theorem: *The points of intersection of the adjacent trisectors of the angles of any triangle are the vertices of an equilateral triangle.*

```
frame(-1,-0.5,7,4.5) A B C triangle a =
angle(B,A,C) b = angle(C,B,A) c = angle(A,C,B)
ab = angle(vector(A,B)) bc = angle(vector(B,C))
ca = angle(vector(C,A)) l1 = line(A,(ab + a/3):)
l2 = line(A,(ab + 2*a/3):) l3 = line(B,(bc +
b/3):) l4 = line(B,(bc + 2*b/3):) l5 = line(C,(ca
+ c/3):) l6 = line(C,(ca + 2*c/3):) D =
```

¹ It was formerly named **euklides**, but this name was already given to other geometry software.

² In */tex-archive/support/eukleides/*.

³ At <http://perso.wanadoo.fr/obrecht/>.

Eukleides: A geometry drawing language

Christian Obrecht

As a mathematics teacher in a French high school, I have to compose a rather large number of documents for my students, containing both text and formulas. In my point of view, *LATeX* is the best tool in such a situation, combining efficiency and high quality. Very often, these documents should be illustrated with geometric figures. I first used the excellent *PSTricks* package to draw them. I didn't want to use WYSIWYG software instead, because I wanted to keep following *LATeX*'s philosophy, that is: What You Mean Is What You Get. Unfortunately, *PSTricks* isn't designed for geometry at all and is rather inappropriate in many situations.

One night, I wanted to draw a triangle with an inscribed circle, so I had to compute by hand the coordinates of the center and the radius of this circle, which is quite boring. During these calculations, I realized that they could easily be done by a computer, and that gave me the idea to create Eukleides, a geometry drawing language. My goal was to make it as close as possible to what mathemat-

```

intersection(11,14) E = intersection(13,16) F =
intersection(12,15) color(lightgray) draw(l1) ;
draw(l2) draw(l3) ; draw(l4) draw(l5) ; draw(l6)
color(black) draw(A,B,C) ; draw(D,E,F)

```

Here is the corresponding code⁴.

```

1 A B C triangle
2 a = angle(B,A,C)
3 b = angle(C,B,A)
4 c = angle(A,C,B)
5 ab = angle(vector(A,B))
6 bc = angle(vector(B,C))
7 ca = angle(vector(C,A))
8 l1 = line(A,(ab + a/3):)
9 l2 = line(A,(ab + 2*a/3):)
10 l3 = line(B,(bc + b/3):)
11 l4 = line(B,(bc + 2*b/3):)
12 l5 = line(C,(ca + c/3):)
13 l6 = line(C,(ca + 2*c/3):)
14 D = intersection(l1,14)
15 E = intersection(l3,16)
16 F = intersection(l2,15)
17 color(lightgray)
18 draw(l1) ; draw(l2)
19 draw(l3) ; draw(l4)
20 draw(l5) ; draw(l6)
21 color(black)
22 draw(A,B,C) ; draw(D,E,F)

```

In Eukleides source code, a line can contain several commands (in that case, they have to be separated by semicolons). Commands are of two kinds: variable assignments and graphical commands. Among variable assignments are single assignments (see lines 2–16) and multiple assignments (line 1). A variable can store a wide variety of objects used in elementary geometry: numbers, vectors, points, lines, segments, circles, conics.

Multiple assignments are used for definitions of polygons and for some intersection determinations. The statement in line 1 defines an optimal scalene triangle such that segment AB is horizontal and 6 cm long. All these characteristics can be modified by adding some optional parameters to the keyword ‘triangle’. For instance ‘A B C triangle(4,5,6)’ would define a triangle ABC such that $AB = 4$ cm, $BC = 5$ cm and $AC = 6$ cm.

If the desired triangle has to be of a specific kind, the simplest way is to replace ‘triangle’ with ‘right’, ‘isosceles’ or ‘equilateral’. For instance ‘A B C right(5,30:,10:)’ would define a triangle ABC with an angle of 30° in A , a right an-

⁴ The numbers at the beginning of each line are not part of it.

gle in B and such that segment AB measures 5 cm and makes an angle of 10° with the horizontal direction. The colon character is used to distinguish angular parameters from others (like lengths).

On lines 2–7, one can see two possible usages of the function ‘angle’. In the first case (lines 2–4), it simply gives the measures of the angles in triangle ABC . In the second case (lines 5–7), it gives the argument of some vectors. As with many functions in Eukleides, ‘angle’ can handle several kinds of arguments.

On lines 8–13 are the definitions of the trisectors of the triangle ABC . Since trisectors (unlike bisectors) aren’t very common objects, there’s no built-in function to define them. The function ‘line’ is used instead. Here, the second argument is the angle that the line makes with the horizontal direction. This is not the only way to define a line: the second argument could have been a point or a vector.

Graphical commands are of two kinds: setting commands (see lines 17 and 21) and drawing commands (see lines 18–20 and 22). To draw an object, one simply has to use the function ‘draw’. This function can take additional arguments in order to modify the aspect of the drawn object (such as ‘dotted’ or ‘dashed’ for lines). On line 22, the arguments of ‘draw’ are a list of points: it’s the way to draw polygons. Since polygons are not considered as specific objects in Eukleides, there’s no need to declare DEF as a triangle before drawing it.

More graphical commands

Usually, a geometric figure doesn’t contain only straight and curved lines, but also letters and some conventional marks (used to make some properties obvious). Below is a classical example of such a figure representing a parallelogram.

```

A B C D parallelogram(5,4,105:) O =
barycenter(A,B,C,D) frame(-2,-1,6,4.5)
draw(A,B,C,D) ; draw(O) draw("A",A,-130:)
draw("B",B,-30:) draw("C",C,50:)
draw("D",D,130:) draw(segment(A,C),dotted)
draw(segment(B,D),dotted) mark(segment(A,O))
mark(segment(O,C)) mark(segment(B,O),cross)
mark(segment(O,D),cross) mark(B,A,D)
mark(D,C,B) mark(C,B,A,double)
mark(A,D,C,double)

```

Here is the corresponding code.

```

1 A B C D parallelogram(5,4,105:)
2 O = barycenter(A,B,C,D)
3 frame(-2,-1,6,4.5)
4 draw(A,B,C,D) ; draw(O)
5 draw("$A$",A,-130:)

```

```

6   draw("$B$",B,-30:)
7   draw("$C$",C,50:)
8   draw("$D$",D,130:)
9   draw(segment(A,C),dotted)
10  draw(segment(B,D),dotted)
11  mark(segment(A,O))
12  mark(segment(O,C))
13  mark(segment(B,O),cross)
14  mark(segment(O,D),cross)
15  mark(B,A,D)
16  mark(D,C,B)
17  mark(C,B,A,double)
18  mark(A,D,C,double)

```

Since this figure is rather simple (from a geometrical point of view) only two assignments are needed. On line 1 is a multiple assignment which defines a parallelogram $ABCD$ such that $AB = 5 \text{ cm}$, $AD = 4 \text{ cm}$ and $\widehat{BAD} = 105^\circ$. On line 2, a single assignment defines O as the center of parallelogram $ABCD$.

Even though Eukleides is designed in order to use as few coordinates as possible, the internal representation of the geometrical objects is based on them. By default, figures are drawn in a frame such that the lower left corner has coordinates $(-2; -2)$ and the upper right corner $(8; 6)$. The function ‘frame’ enables one to change these settings.

As one can see on line 4, the function ‘draw’ is useful to represent single points (the default shape is a dot, but it can also be a square or a cross). This function can also be used to give names to points,⁵ as in lines 5–8. Here, the first argument is a string, the second a point and the third an angular argument specifying the position of the label. This string can contain TeX code⁶ such as mathematical formulas.

On lines 11–18 are the marking commands. It is possible to mark, in various ways, either segments (lines 11–14) or angles (lines 15–18).

A classical locus problem

In some situations, a computer screen can be very useful to teach geometry. For instance, a locus problem becomes much easier if one can see several states of the figure. The program `xeukleides` has been developed for this. At startup, it appears as a text editor. If you type the lines below:

```

1  x interactive(2,.1,0,6,"A",right)
2  A M I equilateral(x)
3  M B J equilateral(6-x)
4  color(lightgray)

```

⁵ Or to put any kind of text in a specific place.

⁶ This code will only be interpreted if you run `eukleides` and `latex`. With `xeukleides` it is displayed verbatim.

```

5  draw(segment(I,J))
6  color(black)
7  draw(A,M,I) ; draw(M,B,J)
8  draw(barycenter(I,J))

```

and press the escape key, the text area will be replaced by a graphical area containing the following figure:

```

frame(-0.5, -0.5, 6.5, 4) x
interactive(2,.1,0,6,"A",right) A M I equilateral(x)
M B J equilateral(6-x) color(lightgray)
draw(segment(I,J)) color(black) draw(A,M,I) ;
draw(M,B,J) draw(barycenter(I,J))

```

If you now press the right arrow key, you’ll see the left triangle becoming bigger and the right one smaller. Pressing the left arrow key performs the opposite transformation. Pressing the escape key again switches back to the text editor.

On line 1 of the source code is an interactive assignment: it allows to modify the value of the numerical variable x (and consequently the figure) by pressing the arrow keys. The first argument is the initial value of x , the second the increment which is added to (subtracted from) x every time the right (left) arrow key is pressed. The third and fourth are the optional lower and upper bound. The fifth argument has to be a string containing a single letter. It indicates the key that has to be pressed before modifying the variable.⁷ This is useful when more than two variables have to be bound to the keyboard. The sixth argument is either ‘right’ or ‘up’. It indicates which pair of arrow keys (right/left or up/down) is bound to the variable.

In an interactive assignment, the initial value can be modified while viewing. If you press the F1 key, the program replaces the original initial value in the source code by the last value of the variable and switches back to the text editor.

The first multiple assignment on line 2 defines an equilateral triangle AMI such that segment AM is horizontal and $x \text{ cm}$ long. The second assignment defines an equilateral triangle MBJ such that segment MB is horizontal and $6-x \text{ cm}$ long. A specific feature of polygonal assignments is used here: if the first variable is already in use (and contains a point) its content remains the same (if not, the variable is set to the origin). This implies that segment AB has a constant length of 6 cm and that M belongs to AB .

⁷ Since the program starts viewing in state “A”, there’s no need here to press this key.

Drawing curves

In elementary geometry, the most usual curves are conics. Eukleides provides a large number of function to define and handle these objects. For less common curves, there's the ‘trace’ command. For instance, the figure below illustrates the geometrical definition of a cubic curve known as the *Witch of Agnesi*.

```
frame(-4,-1,4,3) O = point(0,0) c =
circle(point(0,1),1) l = line(point(0,2),0:)
trace(t,.1,179.9) L = line(O,t:) O M
intersection(L,c) P = intersection(L,l)
point(abscissa(P),ordinate(M)) t = 50 L =
line(O,t:) O M intersection(L,c) P =
intersection(L,l) N =
point(abscissa(P),ordinate(M)) draw(O)
style(dotted) draw(segment(M,N))
draw(segment(P,N)) draw(L) thickness(.5) ;
style(dashed) draw(c) ; draw(l)
```

This curve is obtained by drawing a line from the origin through the dashed circle, then picking the point with the x coordinate of the intersection with the dashed line and the y coordinate of the intersection with the circle. Here is the corresponding code:

```
1   frame(-4,-1,4,3)
2   O = point(0,0)
3   c = circle(point(0,1),1)
4   l = line(point(0,2),0:)
5   trace(t,.1,179.9){
6   L = line(O,t:)
7   O M intersection(L,c)
8   P = intersection(L,l)
9   point(abscissa(P),ordinate(M)))
10  t = 50
11  L = line(O,t:)
12  O M intersection(L,c)
13  P = intersection(L,l)
14  N = point(abscissa(P),ordinate(M))
15  draw(O)
16  style(dotted)
17  draw(segment(M,N))
18  draw(segment(P,N))
19  draw(L)
20  thickness(.5) ; style(dashed)
21  draw(c) ; draw(l)
```

On lines 2–4 we create the objects which are needed to define the curve. Lines 5–9 are related to the ‘trace’ command. They are based on the geometrical definition above. Line 5 tells variable t to scan the numbers between⁸ 0.1 and 179.9. A line-circle intersection can lead to two points, hence in Eukleides a multiple assignment (like the one at line 7) is used to obtain these points. Since lines are implicitly directed, in the present case the first assigned point will always be O and the second, the wanted point. The last line (line 9) contains a point valued expression. This is the point which will be drawn for each value of t .

To draw this curve, it would also be possible to use parametric representation. Nevertheless, in the present case the geometrical definition is more appropriate because the same piece of code can be used again (in lines 11–14) to produce an example of the construction.

The last part (lines 15–21) contains the drawing commands. The setting command ‘style’ changes the default aspect of drawn objects. This may sometimes shorten the code.

Conclusion

In my humble opinion, Eukleides is now mature enough to be considered by TeX users as an effective way to create geometric figures. As a matter of fact, the language is sufficiently powerful to describe almost any figure which can be seen in an elementary geometry textbook.

My aim is now to enhance Eukleides with features such as tests, loops, and user-defined functions. Since I did not anticipate this when I started the project, I'll have to rewrite large parts of the programs. This is a long-term undertaking, so I'll soon stop working on the present versions of *eukleides* and *xeukleides*.

◊ Christian Obrecht
3, impasse Bellevue
89100 PARON
FRANCE
christian.obrecht@wanadoo.fr
http://perso.wanadoo.fr/
obrecht/

⁸ These bounds are chosen in order to avoid 0 and 180, which are invalid and may cause spurious lines to appear.

Book Review

Book review: *T_EX Reference Manual*

Stephen Moye

David Bausum, *T_EX Reference Manual*. Kluwer Academic Publishers, Boston, Dordrecht and London, 2002, ISBN 0-7923-7673-0. \$100.

A new T_EX book!

The arrival on the scene of a new book about T_EX is always an occasion for great joy. I'm only too happy to have another reference that causes me to look at a problem in a new way.

Teaching by example

A useful approach. Bausum's book is particularly useful in that it covers T_EX's primitives. It is therefore of use to anyone who uses any flavor of T_EX. The author asks early on, "Why is T_EX so hard to learn?" He asserts that the reason is twofold: It is a large programming language with 325 primitives, and parts of T_EX are not intuitive. The purpose of the book, I take it, is to address these issues and to make the learning process more efficient.

The author begins by separating T_EX primitives into nineteen "families" such as *The Box Family*, *The Font Family*, *The Paragraph Family* and *The Tables Family*. This is a useful approach for those starting out because it brings a greater sense of structure to the primitives other than their relevance to vertical and horizontal mode.

The next section—the bulk of the book, about 310 pages—is given over to an annotated listing of T_EX primitives, complete with extensive cross references to *The T_EXbook* and examples for virtually every entry. At the head of each entry is a kind of graphical/shorthand overview of the primitive—be sure that you review the first two pages of this section (pp. 25–26) so that you know how to interpret this. The book concludes with three appendices, "Typesetting Verbatim Material", "Working with PostScript Fonts" and "Typesetting Material in Two columns".

Generally speaking the content is very good. I like lots of examples about how things work—to learn by doing. Each of the T_EX primitives is covered in three parts: a description of what the primitive does; examples of how it can be used, accompanied by the output of the examples where appropriate; and finally a commentary to clarify issues that may have been raised by the examples. It

is clear that the *T_EX Reference Manual* is not meant to stand on its own. You will want to have a copy of Knuth's *The T_EXbook* close at hand. Bausum also makes occasional reference to some other of Knuth's books if he thinks the material there explains a given issue better, so you will have to have access to them as well.

Occasionally there is a less-than-ideal turn of phrase. On page 220, Bausum says: "Normally, an output routine has no idea where it is in a document." Hmm... Yes and no: The output routine may not know *exactly* where it is on a given page, but it does know enough to form a decision as to how much of the page it has to fill, and which page it is on—a useful piece of information for formatting that requires different things on even as opposed to odd pages. Still, these failings are generally minor, and, to do the author credit, the appropriate pages in Knuth's books are copiously referenced if there is any question.

One modest grievance I have centers on the clock that adorns the beginning of the discussion of each primitive. The macro for producing it (`\mkclockA`) is not listed in the index. It is only defined in the course of an example centering on the primitive `\special`. So obvious a formatting feature should have been better documented, nor is the PostScript file (`clock.ps`) given. I believe strongly that books about typography should completely elucidate the details of their own creation.

When bad things happen to good books

Given Knuth's exhortation, "Go forth now and create masterpieces of the publishing art,"¹ there are some less than masterful touches in evidence here.

FONTS. The choice of font, Caslon 224, is less than happy in my opinion. First, it is a very idiosyncratic version of Caslon. Second, there are the ligatures. Given the discussion of fonts in Appendix B, I am surprised that no ligatures are in evidence here beyond the usual *fi* and *fl* particularly in view of the fact that the unligatured *ff*, *ffi* and *ffl* are notably odd looking. But then, there is no "expert set" available for Caslon 224 which would have provided the missing ligatures. Surely, Adobe Caslon or Berthold Caslon (both of which have expert sets) would have provided more attractive type.

To T_EX or not to TeX. And then there is the matter of the logotype: T_EX. The author feels that what he calls the "familiar form" (TeX) is less distracting than the formal form (T_EX). Oddly,

¹ *The T_EXbook*, page 303

the formal form is used in the Preface, while the familiar form is used in the rest of the book. First, I think that anyone who is serious about learning *T_EX* had best get used to the formal form in very short order. Indeed, I can think of no serious work about *T_EX*—and typeset using *T_EX*—that does not use the formal form. The use of the familiar form makes the book look as if it were composed using Quark *XPress* rather than *T_EX*. Using ‘*T_EX*’ is just the right thing to do. Second, reverting to a familiar form is no excuse for doing the formal form badly. The half-title, title and back cover have some very unlovely interpretations of the formal form.

Printing. The print quality is less than wonderful. The book was apparently mastered on some sort of laser printer of modest resolution. The resulting hard copy was used to make printing materials in such a way that the type occasionally comes close to breaking up: serifs are degraded, thin strokes tending almost to disappear.

The price tag. I feel compelled to venture the opinion, given the less-than-stellar production values evidenced in this book, that \$100 for it seems excessive—there isn’t even a CD with the macros and examples shown in the book. There is absolutely no doubt in my mind that this is a fine \$30–\$50 book, but \$100 renders it considerably less attractive to a prospective purchaser than it could, and should be.

The bottom line

This is a good reference for people who have a bit of plain *T_EX* under their belts, as the examples mix primitives and plain rather freely. If you can find this book for a reasonable price, buy it because it is a useful and informative book. Despite some failings—some superficial, some not—it is worth having in your reference library, particularly if you use plain *T_EX*, or have to delve into *T_EX*’s innards for any reason.

◊ Stephen Moye
American Mathematical Society
201 Charles Street
Providence, RI 02904
sgm@ams.org

Hints & Tricks

Glisterings

Peter Wilson

Not all that tempts your wand'ring eyes
 And heedless hearts, is lawful prize;
 Nor all, that glisters, gold.

Ode to a Favourite Cat
 THOMAS GRAY

For many years Jeremy Gibbons has edited a very successful column in *TEX* and *TUG NEWS* and *TUGboat* called *Hey — It works!*[3]. I have learnt much from this but apparently not enough to decline when asked to take over the column. On the other hand I have learnt to my cost that the quickest way to get a correct answer to a question on the *comp.text.tex* (*c_tt*) newsgroup is to give an incorrect answer. In order not to sully Jeremy's reputation my first thought was to change the title to *Hey — It might work* but after some consideration the new title is as you see it above — *Glisterings* — implying that there might be some dross among the nuggets.

Corrections, suggestions, and contributions will always be welcome.

Several questions on *c_tt* recently have been related to comparing two words or strings. To my chagrin I gave an incorrect answer to one of the questions, so I'll now try and redeem myself.

If you can meet with triumph and
 disaster
 And treat those two imposters just the
 same...

If —
 RUDYARD KIPLING

Checking for an optional argument

If you are defining a new command that has an optional argument you often need some way of checking whether or not it is present when the macro is called, especially when it should be ignored if it is not present. One convention is to use the kernel \empty macro as the default for the optional argument.

```
\newcommand{\mine}[2][\empty]{%
% if #1 is \empty do nothing else
% do something}
```

To me the obvious way of performing the check was to use *TEX*'s \ifx primitive to compare \empty and the actual value of the argument, as in

```
\newcommand{\testoptarg}[1][\emptyset]{%
  \ifx #1\emptyset
    Optional (#1) unused%
  \else
    Optional (#1) present%
  \fi}
```

If you try this you can get some odd results:

```
\testoptarg Optional () unused
\testoptarg[full] Optional (full) present
\testoptarg[oops] psOptional (oops) unused
```

It was kindly explained to me¹ that `\ifx` checks the following two tokens and in TeX a token is either a command sequence (e.g., `\emptyset`) or a single character, like ‘o’. In the oops example, `\ifx` checks ‘o’ and ‘o’, concludes that they are the same, and hence the strange result. Flipping the token ordering works better:

```
\ifx\emptyset#1
```

Now `\testoptarg` and `\testoptarg[\emptyset]` will report ‘Optional () unused’. Any other call, for example `\testoptarg[]`, will report ‘Optional () present’, and in particular `\testoptarg[oops]` reports ‘Optional (oops) present’.

String comparisons

A more general problem along the same lines is to check if two words, or strings are the same. We can use `\ifx` for this as well. When `\ifx` compares two tokens that are macro names, the result is true if the macros have been defined in the same way, and if their first level replacement texts are the same. So, we define two macros whose replacement texts are the strings, and compare these.

```
\newif\ifsame
\newcommand{\strcfstr}[2]{%
  \samefalse
  \begingroup
  \def\1{#1}\def\2{#2}%
  \ifx\1\2\endgroup \sametrue
  \else \endgroup
  \fi}
```

The two arguments to `\strcfstr`² are the strings to be tested. `\ifsame` is set true if the two strings match character to character. If the arguments are macro names it checks the characters in the names, not their definitions. If there are any spaces in the arguments, each group is reduced to a single space before the strings are compared. `\strcfstr{}{}` sets `\ifsame` false but `\strcfstr{}{}{}` sets it true.

```
\newcommand{\StrCfStr}[2]{%
```

¹ By, among others, Donald Arseneau, Michael Downes and Stephan Lemke.

² The `cf` used in the names of macros is the abbreviation `cf` (from the Latin *confer* = compare).

```
\lowercase{\strcfstr{#1}{#2}}}
```

The `\StrCfStr` macro performs a case insensitive test on two strings. For example, it will set `\ifsame` true for any of the pairs (abc, abc), (abc, Abc), (abc, aBc), and so on. It uses `\lowercase` to convert any uppercase letter to a lowercase letter so all the letters will be lowercase at the time `\strcfstr` does the checking. This will not work if the arguments include differently cased macro names as `\lowercase` does not touch those.

The `\strcfstr` and `\StrCfStr` macros have provided all the string testing that I have needed, but I'll show a couple of extensions. One thing is that `\strcfstr` relies on `\def` which is not expandable so, for example, it cannot be used in an `\edef`. Both Victor Eijkhout [2, section 13.8.7] and David Kastrup [4] have presented solutions for this. The other is that you may want to check if a macro expands to a particular string. David's expandable macro also provides a solution for this and Michael Downes[1] gives a somewhat different method using `\expandafter`.

We can use `\strcfstr` as the basis for the macro to string comparison, by using `\expandafter`.

```
\newcommand{\macrocfstr}[2]{%
  \expandafter\strcfstr\expandafter{#1}{#2}}
```

The first argument to `\macrocfstr` is either a string or a macro that is expected to expand to a string. The second argument is the test string.

We can also do a case insensitive test by using

```
\newcommand{\MacroCfStr}[2]{%
  \lowercase{\macrocfstr{#1}{#2}}}
```

The `\charscfchars` expandable macro below is based on Victor's code. It is tricky because it uses recursion to perform pairwise comparisons of the individual characters in its two arguments, and it requires two supporting macros.

```
\catcode`\^G=11 % make a letter
\newcommand{\charscfchars}[2]{%
  \IfAllChars#1^G#2^G\theSame}
```

`\charscfchars` adds a character at the end of its arguments to mark the ends of the strings. Victor used \$ as the marker which meant that neither argument could include \$ among the characters. I chose to use `^G` (TeX's notation for the ASCII BEL control character, which is normally invalid TeX). The `\catcode` changes first make `^G` appear to be a letter and then at the end of the macro definitions it is set back to its normal invalid state.

The next macro, which is presented with some interspersed commentary, does most of the work.

```
\def\IfAllChars#1#2\Are#3#4\TheSame{%
  \if#1^G\if#3^G\sametrue
  \else\samefalse\fi}
```

The macro takes two pairs of arguments that are delimited by the tokens `\Are` and `\TheSame`. The first pair of arguments are for the first string under test and the second pair for the other string. More specifically, #1 will be the first character in the first string and #2 contains the remaining characters (including the `^G` marker), and similarly for #3 and #4. If the ends of both strings have been reached, then the strings are the same, but if only the end of the first string has been reached, the strings are different. If we are not at the end of the first string there is more work to be done.

```
\else\if#1#3\IfRest#2\TheSame#4\else
  \samefalse\fi\fi}
```

If the corresponding characters in the two strings are the same then the rest of the character pairs must be checked, otherwise the characters don't match and we are done.

The last of the macros takes three arguments which are delimited by the tokens `\TheSame`, `\else`, and `\fi\fi`. The first two arguments are strings to be compared, and it throws away the third.

```
\def\IfRest#1\TheSame#2\else#3\fi\fi{%
  \fi\fi \IfAllChars#1\Are#2\TheSame}
\catcode`\\=15 % return to invalid
```

This macro simply calls `\IfAllChars...` to compare the strings.

`\charscfchars` can be used as a basis for case insensitive and macro to string comparisons exactly like `\strcfstr`.

Apart from `\charscfchars` being expandable while `\strcfstr` is not, it also ignores all space characters while `\strcfstr` does not. For example, `\charscfchars{ab}{a\b}` thinks that the arguments are identical but they will be reported as different if `\strcfstr{ab}{a\b}` is used.

References

- [1] Michael J. Downes. Re: catcodes for jobname macro — stupid question. Post to `comp.text.tex` newsgroup, 25 April 2001.
- [2] Victor Eijkhout. *TeX by Topic, A TeXnician's Reference*. Addison-Wesley, 1991. ISBN 0-201-56882-9. (Available at <http://www.eijkhout.net/tbt/>).
- [3] Jeremy Gibbons. Hey — it works! *TeX and TUG NEWS*, 2(2):7–11, April 1993.
- [4] David Kastrup. Completely expandible string comparison. Post to `comp.text.tex` newsgroup, 3 September 2002.

◊ Peter Wilson
 18912 8th Ave. SW
 Normandy Park, WA 98166 USA
`peter.r.wilson@boeing.com`



The Treasure Chest

Packages posted to CTAN

The vast number of available packages is at once a wonderful resource and strength of the *TEX* community, a veritable embarrassment of riches as well as an awkward political minefield and urban turf war (divisions between (Plain) *TEX* and *LATEX* and other macro packages). Despite extraordinary efforts on the part of volunteers like Graham Williams, the CTAN team and participants in *comp.text.tex*, and most especially Robin Fairbairns, maintainer of the UK *TEX* FAQ, available at <http://www.tex.ac.uk/faq>, many users find it difficult to find the appropriate package for their needs.

This is a chronological list of packages posted to CTAN between January and June 2001 with descriptive text pulled from the announcement or researched at need and edited for brevity—however, all errors are mine. Packages are in alphabetic order and are listed only in the last month they were updated. Individual files / partial uploads are listed under their own name if so uploaded. If not otherwise noted, packages are in *macros/latex/contrib/supported/*. Subdirectories (e.g. *foo*) of *macros/latex/contrib* are listed as *.../foo* to save space.

This is my second of three columns for “The Treasure Chest”. Mark LaPlante is scheduled to do the next column (for July–December 2001); my final column will cover the beginning of 2002; and Mark’s tour of duty will then continue without further interference from yours truly. Hopefully this column and those which follow will help to make CTAN a more accessible resource to the *TEX* community.

Corrections and suggestions are welcome.

January 2001

attachfile Attach arbitrary files to PDF documents for later extraction. Also adds the ability to modify the file icon.

circuit_macros in **graphics**

(V.5.0) A set of macros for drawing high-quality electric circuit diagrams containing fundamental elements, amplifiers, transistors, and basic logic gates. Several tools and examples for other types of diagrams are also included. More robust NOT_gate, Function pmod(), macro shade(), etc. Examples are now in *Xfig* version 3.2 format, web-based examples and updated links, new, more robust macros, editorial changes to the manual.

cmtiup in fonts/cm

Replacement of `cmti*` fonts by `cmtiup*` simplifies typesetting of articles where otherwise author or editor have to use additional commands in italic text with formulas.

CWEBbin in web/c_cpp

(V.3.63) A set of change files (to be applied with the TIE processor) that make the original sources usable with ANSI-C/C++ compilers on UNIX/Linux, MS Windows, and Amiga. Extra functionality, like macros, and macros for international documentation of CWEB programs, is introduced.

dice3d in fonts/dice

METAFONT source for 2D and 3D pictures of dice.

EC in nonfree/systems/win32/bakoma/fonts

EC fonts in ATM compatible Type 1 font format intended for installing under BaKoMa \TeX 3 and later.

Eplain in macros/eplain

V.2.8.1 of Eplain, a macro package that extends the definitions of plain \TeX .

hypernat.sty in .../supported/misc

(V.1.0a) Makes `hyperref` and `natbib` with options `numbers` and `sort&compress` work together. Thus the citations (e.g. [3,2,1]) will be compressed to [1–3], where the ‘1’ and the ‘3’ are (color-)linked to the bibliography.

hyphenat This package can be used to disable all hyphenation in a document or just in selected text. It also enables hyphenation of words with analphabetic characters (e.g., that include underscores), and hyphenation in monospaced fonts. V.2.3a fixes a double hyphen problem with some fonts.

IEEEtran_v13.cls in .../supported/IEEEtran

This is a new beta test version of the class for authors of papers in journals published by the Institute for Electrical and Electronics Engineers (IEEE).

jurabib Supports various forms of short and long citations—now more flexible and no longer just for German law students. Changes and improvements in v.0.5e too numerous to mention.

Kto8 in systems/mac

Converts the strangely ASCII-encoded text files often used by \TeX into true 8-bit text files. Presently a freeware program realized for Macintosh OS as a tiny drag-and-drop utility.

LATEX2E.BKZ in nonfree/systems/win32/bakoma/dst

This module includes updated $\text{\LaTeX}2\epsilon$ with required packages, $\mathcal{AM}\mathcal{S}$ - \LaTeX , and HyperRef packages.

LH in nonfree/systems/win32/bakoma/fonts

LH fonts in ATM compatible Type 1 font format intended for installing under BaKoMa \TeX 3 and later.

MiKTeX in systems/win32/miktex/2.0

MiKTeX is a free \TeX distribution for Windows.

minutes.sty in .../supported/minutes

Writing minutes, v.1.6 offers: new votes, new fields for title (location, guest) and solved some bugs.

mfpic in support/mfpic

(V.0.1) Perl script interface for making pictures with `mfpic`.

ps2eps in support

A tool to produce EPS/EPSF files from usual one-paged PostScript documents. New v.1.28 improves the calculation of bounding boxes. Requirements: Perl, Ghostscript and an ANSI-C compiler if your platform is not Linux, Solaris, Digital UNIX or Windows 2000/9x/NT (binaries included).

sepnum.sty in .../supported/misc

This package provides a means of formatting numbers with (a) a decimal separator different than the default ‘.’ and (b) a separator (default ‘,’) every three digits. All separators are user-definable; the macro implementation is fully expandable and can therefore be used in places where problems occur with fragile commands.

titlesec Essentially a replacement—partial or total—for the \LaTeX macros related to sections; namely, titles, headers, and contents (upgrade to v.2.4). This version contains the incremental updates in previous releases and bug fixes.

ttf2pt1-3.3.2 in fonts/utilities/ttf2pt1

System for converting TrueType fonts to Type 1 and installing them.

TXFONTS.BKZ in nonfree/systems/win32/bakoma/fonts

TX Fonts (3.1) and PX Fonts (1.0) in ATM compatible Type 1 font format intended for installing under BaKoMa \TeX 3 and later.

WinShell in systems/win32/winshell

Beta version of WinShell, a graphical user interface for easily working with \TeX . It is *not* a \TeX system so requires one to have a system such as Mik \TeX or \TeX Live installed.

xtem in support/xtem_texmenu/xtem.v8

Provides for a simple and comfortable graphical user interface (GUI) to control \TeX / \LaTeX . Runs with new Tcl/Tk version 8. First version which does not need `TclX`. Select boxes in setting menus are now expandable, modifications of logfile analyze; warnings can now be displayed and editor options can now be modified.

ziffer in .../supported/misc

Formats numbers with the correct German spacing (even in math mode).

February 2001

abstract Gives you control over abstracts, and in particular provides for a one column abstract in a two column paper. V.1.1 adds an option for a run-in heading.

ae in fonts/ae

This package is a set of virtual fonts for creating

PDF files with T1-encoded CMR fonts. New in v.1.3 adds support for the slides fonts.

ASCII-Cyrillic in *language/ascii-cyrillic*

A new system for dealing precisely with Cyrillic languages using no more than an ASCII keyboard and screen font.

babel in *macros/latex/required/babel*

(V.3.7) Very much like the later releases of **babel** 3.6k and later. The concept of language attributes is new in this release and support for a number of languages has been added.

catdvi in *dviware*

(V.0.11) The program is a .dvi-to-plain text translator capable of generating ASCII, Latin-1 and UTF-8 (Unicode) output. It aims to become a superior replacement for the **dvi2tty** utility. The previous version is usable for previewing (on character-cell displays) .dvi files that contain mostly linear text. The new version includes a much improved layout algorithm which can generate a fairly pleasantly readable text file from a multicolumn .dvi file.

cd-cover in .../other

Class for typesetting various forms of CD covers.

chemarrow in .../other

Macro and font provide new arrows for chemical reaction schemes. Easy to use and aesthetically more pleasing.

chngpage Provides commands to change the page layout in the middle of a document, and to robustly check for typesetting on odd or even pages. V.1.1c fixes odd/even page check problem when used with **calc** or **hyperref**.

epsdice Scalable dice font, accepts counter values as arguments; v.1.1 works with pdfL^AT_EX.

epstopdf.sty in .../supported/oberdiek

Adds support for handling images in eps format to **graphic{s,x}** with option **pdftex**.

euro-ce in *fonts*

METAFONT source of the official Euro currency symbol and CE logo including variants, in both solid and outline.

GenMPage in .../supported/genmpage

Generalizes L^AT_EX's minipages. Keyval options and styles can be used to determine their appearance in an easy and consistent way. V.0.3 introduces options for paragraph indentation and vertical alignment with respect to the visual top and bottom margins.

hvmaths.ins in .../supported/psnfssx

Install scripts can be processed with plain or IniTeX.

JavaBib in *biblio/bibtex/utils*

GUI BibTeX reference manager written in Java. Beyond easy reference entry, JavaBib can search a database, and paste citations to the system clipboard.

jPicEdt in *graphics/jpicedt*

Java program that allows you to draw pictures from a menu-driven interface, using L^AT_EX's **picture** environment or **epic/epic** macro packages.

listings Source code printer for L^AT_EX. This update adds support for Visual Basic and removes some bugs.

ly1.txt in .../supported/psnfssx

Fixes a misleading typo.

nomentbl in .../other

This package is a redefinition of the **nomencl** package. Instead of the usual **makeindex** style it uses a **longtable** environment to put the symbol list in three columns: symbol, description and unit.

pdflscape.sty in .../supported/oberdiek

Adds PDF support to the environment **landscape** of package **lscape** by setting the **page** attribute **/Rotate**. Also works with **dvips**.

pdftricks Helps to include **pstricks** code in a document to be processed by pdfL^AT_EX.

prelim2e in .../supported/ms

(V.1.23) Package to mark preliminary versions of a document. Adds **danish** and **italian** options.

psnfssx Cleaned up and reorganized. This directory is the home of packages that complement the basic PSNFSS distribution, primarily to support commercial PostScript fonts. The following subdirectories are provided: **adobe**, **lucidabr**, **mathtime**, **hvmath**, **tmmath**, **ly1**, **8r** and **obsolete**.

rfc2bib.awk in *biblio/bibtex/utils*

gAWK script to automatically generate BibTeX entries from IETF RFCs. The script has been updated to better handle RFCs with periods in the title and to deal with what appears to have been a change in the format of the date for *just* RFC768.

sviewer351.exe in *systems/win32/scientificviewer*

Scientific Viewer 3.51 is a free program for reading and printing (read-only) documents created with Scientific Notebook, Scientific Word or Scientific WorkPlace by MacKichan Software, Inc. It can also be used to view many native L^AT_EX documents.

tmmaths.ins in .../supported/psnfssx

Install scripts can be processed with plain or IniTeX.

wp-conv.zip in *help*

Updated version of FAQ pages on converters between (I^A)T_EX and wordprocessors.

March 2001

alg Added support for German. Fixed typo in the French language support. Now uses the language selected for the **babel** package and has more flexibility in handling **marginwidth**.

allrunes in *fonts*

Almost all runes ever used in Europe, including not only the main forms, but most variant forms.

AlProTeX.sty in *web/protex*

Minor modifications to this literate programming system, for compatibility with TeX4ht.

appendix Provides various ways of formatting the titles of appendices. Also (sub)appendices environments are provided that can be used for per chapter/section appendices.

bophook (V.0.2) Hook for adding material at the beginning of each page. New version adds partial support for **hyperref**.

boundbox.sty in **.../supported/sttools**
Calculate TeX Bounding Box in points.

braille.sty in **.../supported/braille**
Update of package for typesetting braille.

bundledoc A post-processor for the **snapshot** package that bundles together all the classes, packages, and files needed to build a given document. Ships with configuration files for teTeX/Linux and MikTeX/WinNT.

ccaption Provides continuation captions, unnumbered captions and legends, captions outside **float** environments, bilingual captions, etc. It also enables the definition of new **float** environments and their captions. Tools are provided for defining your own captioning styles. V.3.0 provides the same tools as **tocloft** for styling List of floats.

chbibref in **.../supported/misc**
Provides a class-independent means of changing the Bibliography/References title.

chngcntr in **.../supported/misc**
Provides commands to change the resetting of counters (i.e., it provides wrappers for **\@addtoreset** and **\@removefromreset**).

compactbib.sty in **.../supported/compactbib**
Enables multiple bibliographies with different names and continuous numbering.

comprehensive in **info/symbols**
Major update of this list of well over 2,000 symbols available for use in LATEX. Changes include: T1 fonts used only where absolutely necessary, the addition of a README file and an ASCII symbol list (SYMLIST), and a prebuilt PostScript version. Symbol tables have been categorized into “text”, “math”, “science/technology”, and “other” symbols.

CurVe in **.../supported/curve**
A class for making CVs (in distinct versions). Provides commands to create rubrics, entries in these rubrics, etc.

cuted.sty, midfloat.sty in **.../supported/sttools**
Mixing **onecolumn** and **twocolumn** modes at any place on a page.

cv A style for a curriculum vitae and an example CV.

dviinfo in **dviware**
Extracts and displays information regarding a **dvi** file.

eulervm in **fonts**
(V.2.6) This is a set of *virtual* math fonts, based on Euler and CM. Included is a LATEX package, which makes them easy to use, particularly in conjunction

with Type 1 PostScript text fonts. Should work with the latest **amsmath.sty** and now has an option to load the fonts at 95% of their nominal size.

excel2latex in **support**

Exports the current selection of a Microsoft Excel spreadsheet as a file, which can be included in an existing document. Nearly all formatings are supported (bold, italic, border lines, multicolumn cells, etc.). Now works with Excel 97.

fink This package looks over your shoulder and keeps track of files **\input** or **\included** in your document and provides a macro to access the name of the file currently being processed.

fixme Provides ‘fixme’ notes in draft documents, includes support for AUC-Tex.

floatpag.sty in **.../supported/sttools**
Different pagestyles for text and float pages.

flushend.sty in **.../supported/sttools**
Columns balancing at last page.

french in **language/french**

Light version has a maximum of automatic features: translation, layout, microtypography, etc., but no specific commands are available. Footnotes should no longer generate an “undefined!” error message.

FrenchPourWin in **nonfree/language/french**
Windows distribution of French Pro V.5.02.

fullpict.sty in **.../supported/fullpict**
Enhances the previous environment **scalepicture** in that it works properly inside **tabular** and **minipage** environments, and also chooses appropriate font size for the scale of the picture.

hanging Facilitates typesetting of hanging paragraphs. It also enables typesetting with hanging punctuation. V.1.2 adds option for less aggressive hanging punctuation.

iso-8859-2.xmt in **macros/xmltex/base**
Encoding support file.

KTeXShell in **systems/unix**

A graphical user interface to TeX, LATEX, and related programs, running on Linux/UNIX with KDE. It is not another WYSIWYG approach. Instead it provides a document development interface where you can define your files, edit, compose, view, and print them with a mouseclick. V.0.4.0 updates to compile under KDE2.

lcg in **.../other/lcg**

Writes random numbers (integers) to a counter via a linear congruential generator (Schrage’s method). Corrected docs and implements a key “quiet” to suppress the output to the screen and the log-file.

marginal.sty in **.../supported/sttools**

Enlarge free and show lost marginal inserts.

MNRAS in **.../supported/mnras**

Class file that enables authors to produce papers for submission to *Monthly Notices of the Royal Astronomical Society* and to produce references in a suitable format.

MPEdit in *graphics/metapost/tools*

METAPOST text editor for Windows 98, 95, NT 4, and 5, and ME. Provides the main possibilities on creation and editing of METAPOST files.

numprint Prints numbers with a separator every three digits and converts numbers given as 123e456 to $123 \cdot 10^{456}$. If an optional argument is given it is printed upright as a unit.

paralist This style file provides some new list environments. Itemized and enumerated lists can be typeset within paragraphs, as paragraphs and in a compact version. Most environments have optional arguments to format the labels. Additionally, the L^AT_EX environments **itemize** and **enumerate** can be extended to use a similar optional argument.

polynom Implements macros for manipulating polynomials, for example it can typeset long polynomial divisions.

presfull.pdf in .../supported/sttools
“Inside L^AT_EX 2_< kernel” by Sigitas Tolusis.

PSTricks in *graphics/pstricks*

New versions of **pst-gr3d** and **pst-poly** (to draw for three dimensional grids and to draw polygons) and a new contribution **pst-lens** to simulate the effect of a lens.

px_patch1.zip in *fonts/pxfonts*
Fixes an encoding mistake in Math Italic fonts.

sc3demo.tex in .../supported/sidecap
Small example for the package **sidecap**.

stabular.sty in .../supported/sttools
Multipage tabular.

stfloats.sty in .../supported/sttools
Floating baselineskip, footnotes below the floats, dblfloats at bottom.

TeX4ht in *support*

An enhanced version of a system for translating (L^A)T_EX sources into hypertext.

texor111.exe/txt in *systems/win32/util*
Free Windows front-end with buttons which can be assigned to run specific programs.

TeXProject.sty in *graphics/dratex*
An enhanced version of a (L^A)T_EX-based project management system.

TeXshade in .../supported/texshade
A macro package for setting shaded nucleotide and protein alignments. It can process multiple sequence alignments in the .MSF and .ALN file formats. Update fixes some bugs and offers new commands for the fine adjustment of feature lines and allows one to add a caption to the alignment.

texsort.sty in .../supported/sttools
Sort/compress numerical lists.

TeXtopo in .../supported/textopo
Plots topology data of membrane proteins derived from PHD predictions, from SwissProt database files or from manually entered data, and transmembrane

domains as seen from above or beneath the cell membrane. Fully compatible with **TeXshade**—this allows one to apply calculated shading based on sequence conservation and functional aspects of the residue sidechains.

tfmpk in *fonts/utilities*

Font viewer for T_EX tfm/pk fonts.

thumbpdf in *support*

V.2.9 provides support for thumbnail previews in PDF files.

titling Provides control over the typesetting of the \maketitle command, and makes the \title, \author and \date information permanently available. V.2.1 adds support for new titling elements, additional controls for the layout of thanks text and a method for centering on the physical page.

tmview in *dviware*

(V.01.03) Fixes a number of bugs and adds some limited support for color specials.

ttf2tex in *support*

Bash script which will create all files neccessary to use TrueType fonts with teT_EX.

tx_patch1.zip in *fonts/txfonts*

Fixes an encoding mistake in math italic fonts.

varindex Provides a convenient front-end for the \index command. For example, it allows generation of multiple index entries in almost any form by a single command. Extremely customizable. Works with all versions of L^AT_EX and most other T_EX formats, too.

windows-1250.xmt in *macros/xmltex/base*
Encoding support file.

April 2001

aeguill Update to v.1.01 to work properly with the **frenchle** package. Provides several kinds of guillemets which are useful with the AE font which lacks guillemets.

amsldoc in *info/italian/amsldoc*

Italian translation, “Manuale utente per il pacchetto **amsmath**.”

amsthdoc.it in *info/italian/amsthdoc*

Italian translation of **amsthdoc**, “Utilizzo del pacchetto **amsthm**.”

authblk in .../other/preprint

Allows footnote-style printing of author and affiliation.

balance in .../other/preprint

Balances the columns of two-column mode.

dashrule Draws a huge variety of dashed rules in L^AT_EX, with parameters for the pattern of dash segments and the space between.

clock Graphical clocks (with a classical 12h dial and two hands) and text clocks (in 24h format) which can show an arbitrary or system time with user expandable appearance.

combine (V.0.42) Bundles individual documents into a single document, such as when preparing a conference proceedings. The auxiliary **combinet** package puts the titles and authors into the main document's Table of Contents. Update works with **fancyhdr** and fixes a pagebreaking problem.

diagnose Provides tools to allow package authors, system administrators or users to determine if everything a given package needs is installed.

diffs-m_it in **info/italian/amsmath**
Diff files for Italian — “Vari files AMS”.

dingbats in **fonts**
Provides L^AT_EX names for some of the characters in the **dingbat.mf** and **ark10.mf** fonts, including decorations, pointing hands, pencil, anchor, etc.

docindex in **macros/latex/exptl/xdoc**
Reimplementation of the doc mechanisms for sorting and formatting a sorted index with easier configuration.

dpffloat Assists in formatting double-page figures or tables, as two consecutive elements. Ensures that the first of the pair is on a left-hand page.

eps_anleitung.html in **info/german/grafik**
Describes how to create **eps** files under Windows.

eqlparbox Allows one to define tags for **\eqlparboxes** — all such with the same tag are set to the same width.

examdesign Typesetting of exams (incl. matching and multiple choice) and answer keys.

figcaps in **.../other/preprint**
Suppresses printing of figures and puts all captions and tables at the end of a manuscript (as is required by some submission guidelines).

fmp Provides a means to include Functional MetaPost, a Haskell front end to the METAPOST language, into L^AT_EX. Includes documentation in PDF format.

Fontmap.cmr in **fonts/cm/ps-type1/contrib**
Makes certain fonts distributed with **tetex** available to **ghostscript**, including **tx** and **pxfonts**. Update for MarVoSym name change.

formular Macros for typesetting forms (formular documents). Supports blank fields to be written in, or predefined contents.

fullpage in **.../other/preprint**
Sets all margins to be either 1 inch or 1.5 cm, and specifies the page style.

itfancyhdr in **info/italian/fancyhdr**
Italian translation of the **fancyhdr** documentation, “Layout di pagina in L^AT_EX”.

import.sty in **.../supported/misc**
Allow input (with variations) of a file with its own inputs from another directory.

itlshort in **info/lshort/italian**
Italian translation of *The Not So Short Guide to L^AT_EX — Una (mica tanto) breve introduzione a L^AT_EX 2_E*.

koma-script Reimplementation of the L^AT_EX classes (**article**, **report**, **book**, **letter**), “implementing European rules of typography and paper formats” as documented by Tschichold in *Selected Papers on Book Design and Typography*.

manjutex in **language/manju**
Package using T_EX or L^AT_EX 2_E and METAFONT to typeset Manju, a language of North East Asia, belonging to the Tungusic branch of the Altaic languages.

MiKTeX-WinEdt-TrueType-Anleitung in **info/german**
Describes the co-operation of MiKTeX with WinEdt, and includes a German translation of Damir Rakityansky's “Using TrueType fonts with T_EX (L^AT_EX) and pdfT_EX (pdfL^AT_EX)”.

multibbl in **nonfree/macros/latex/contrib/supported**
Facilitates the inclusion of bibliographies written in different scripts, e.g., Greek and Hebrew.

newlfm Integrates the **letter** class with **fancyhdr** and **geometry** to produce letters, (standard or business) memos and faxes. Supports stationery, a database for addresses, other languages, tables, figures, Avery address labels and has a full manual.

noTeX in **biblio/bibtex/utils**
A BibT_EX style that outputs HTML. It can be used to automatically generate bibliographies for the web.

pdfpages Package to enable the inclusion of pages of external PDF documents. V.0.1i enables hypertext features.

pkfix in **support/pkfix**
Attempts to replace bitmap fonts in dvips-generated PostScript files with Type 1 outline fonts.

pl-mf in **language/polish**
METAFONT source for Polish version of Computer Modern.

poligraf in **macros/generic/TeX-PS**
(V.2.0) Robust set of macros for prepress (CMYK separations, cropmarks, color or grayscale bars, mirror print). Works with Plain and L^AT_EX; includes sample files and docs in English and Polish.

srcctx Inserts source specials in .dvi files, enabling switching from a particular point in a previewer which supports them (e.g., Yap or xdvi-22.38) to the matching source in an editor which supports external direction to a specific point.

sublabel in **.../other/preprint**
Enables subnumbering (4a, 4b, 4c...) of even user-defined counters.

texdoctk in **systems/unix/teTeX/1.0/contrib**
(V.0.5.1) A Perl/Tk-based GUI for easy access to package documentation. Bugfix update allows viewing of compressed files and other enhancements (see the README).

textpos Places boxes (containing text, or graphics, or a table, etc.) at absolute positions on the page. Version 1.1e corrects a spacing bug in 1.1d.

tocbibind A package to add document elements like a bibliography or an index to the Table of Contents. The “List of ...” headings can also be put into the ToC. V.1.5 no longer needs `stdclsv` and fixes problem when used with `hyperref`.

tocloft The package provides control over the typography of the Table of Contents, List of Figures, and List of Tables. V.2.2 updated to work with `hyperref`.

ua.dic in `systems/win32/winedt/dict`
Ukrainian dictionary for WinEdt.

uhrzeit Time of day and ranges of time in German and international formats, including ancient formats (minutes superscripted and underlined).

uk-tex-faq in `help`
Moved from `usergroups/uktug/faq`.

upquote Modifies `\verb` and `\verb+at+` so that single quotes are vertical/straight, not curly.

xdoc2 in `macros/latex/exptl/xdoc`

Second prototype for the hypothetical `xdoc` package. Reimplements some of the features found in the standard L^AT_EX `doc` package. Additionally provides support for defining new commands similar to `\DescribeMacro` and new environments similar to the macro environment, for two-sided document layouts, for external cross-referencing, for making index entries for invisible characters, and for optionally ignoring certain prefixes (such as `@` and `@@`) in macro names when sorting them.

May 2001

dehyphn.tex in `language/hyphenation`

Version R31 of the “New German” hyphenation patterns. Fixes two bugs.

diagxy in `macros/generic/barr`

Front end for `xypic` to draw commutative diagrams.

everyshi in `.../supported/ms`

L^AT_EX package which provides hooks into `shipout`. V.3.0 update adds `\AtNextShipout` command.

fixmath in `.../supported/was`

A package to provide italic uppercase Greek letters in math and an additional bold italic math alphabet to be used with Computer Modern.

hvmath in `.../supported/psnfssx`

V.1.2b bug-fix update of this package for using MicroPress’ HV-Math (Helvetica Math) fonts.

icomma in `.../supported/was`

An ‘intelligent’ comma, which yields proper spacing in decimal numbers as well as in mathematical expressions which does not depend on a specific encoding, and works with decimal numbers of arbitrary length.

isodate (V.2.03) Tunes the output format of the `\today` command providing `\isodate`, `\numdate`, `\shortdate`, `\TeXdate` and `\origdate` to print a date argument using the actual date format (German (old and new rules; Austrian), US English,

French, Danish, and Norwegian) for output. Compatible with `bibgerm` style file.

itgrfguide in `info/italian`

Italian translation of the documentation for the `graphics` bundle.

itpfguide in `info/italian/psfrag`

Italian translation (“Il sistema PSfrag”) of the documentation for `PSfrag`.

jlshort in `info/lshort/japanese`

Japanese translation of *The not so short introduction to L^AT_EX 2_E* by Tobias Oetiker, version 3.13.

lettre Designed to write letters in French, German, or English. Configurable, through a number of dimensions and macros; these may be grouped in an “institute package” called at initialization. Includes documentation (in French) and examples.

mathlig in `macros/generic`

Multicharacter ligatures in Math Mode, which can make for much more readable source, e.g., `->` could expand to `\rightarrow`, without compromising the normal math mode use of `-` and `>`.

mdvi in `dviware`

Previewer for `dvi` files which supports a wide variety of font formats (PK, GF, VF/OVF, Type 1 and TrueType).

memoir in `macros/latex/exptl`

Initial (alpha) release of Peter Wilson’s flexible L^AT_EX `documentclass` for typesetting of books such as novels, biographies, histories, etc., with options for trim marks, draft (double-spaced, ragged right, no hyphenation, typewriter font) appearance, various sizes and much more. *Note:* This package has moved out of alpha and into a highly refined state and is now in `memoir`.

mftinc Include pretty-printed Metafont source in a L^AT_EX document.

mtfonts.fdd in `.../supported/psnfssx/mathtime`

Source for the `.fd` files for Y&Y’s MathTime fonts. All fontnames changed to lowercase, to match the files as now distributed by Y&Y (cf. the L^AT_EX bug report `psnfss/3001`).

newcommand.py in `support/newcommand`

Python program to automatically generate L^AT_EX macro definitions for macros which require powerful argument processing.

oesch in `fonts/oesch`

METAFONT script font (partial T1 encoding) for Austrian School Writing in 10 and 20 point sizes.

pdftex.def in `macros/pdftex/graphics`

(V.0.03g) Update to handle zero values for `paperwidth` and `paperheight`.

petri-nets in `macros/generic`

Set of TeX/L^AT_EX packages for drawing Petri-nets (and related models) in PostScript documents, with macros related to PBC, M-nets, etc.

psnfss-beta in `macros/latex/required`

Public test release of the new version 8.2. Integrates the `mathpazo` package into `psnfss`.

snapshot Provides a snapshot of the current processing context external dependencies of the document insofar as it can be determined from inside L^AT_EX.

toolbox (V.2.1) Macros for various tasks often needed in T_EX programming.

ukrhyp in *language/hyphenation*
Update.

upgreek in *.../supported/was*
A package to provide the upright Greek letters from the Euler or Adobe Symbol fonts as additional math symbols, with proper scaling in super- and subscripts.

varindex Customizable and convenient front end for the \index command. Version 2.2 fixes placeholder replacement with commas, adds new variables to control the output, and more. Requires **toolbox** 2.1.

wp2latex in *support*
Conversion from Wordperfect 6–8 into L^AT_EX. Update adds ability to extract WPG to PostScript.

June 2001

abstract Gives you control over abstracts, and in particular provides for a one column abstract in a two column paper.

bakoma in *systems/win32*
Upgrade to BaKoMa T_EX system to v.3.50.

BibTeXMng in *biblio/bibtex/utils*
(V.2.0) Easy to use bibliographic software for Windows. Combines online searching, reference management, bibliography making, and information sharing into a single user-friendly environment. It was written to be used with L^AT_EX, using BibTeX. Update improves interface, provides for HTML bibliography and bug fixes.

bluesky/pfb in *fonts/cm/ps-type1*
Updated to fix a typo in **eexec** routine in the outline font binaries which caused problems when previewing with Acrobat 5.0.

cbgreek.zip in *fonts/greek/cb/mf*
Updated version of CB Greek fonts.

chemarr.sty in *.../supported/oberdiek*
(V.1.1) Chemical reaction arrows with the possibility to put text above and below. Requires **amsmath**.

emtexTDS/fix007 in *systems/os2/emtex-contrib*
Fix pack for emT_EX/TDS 0.55 (OS/2).

esdiff Typesets derivatives, partial derivatives, multiple derivatives.

footmisc Updated version with improved support for L^AT_EX code, better hiding of debugging information and improved per-page algorithm (tested on documents in the hundreds of pages).

gtext-letter in *support*
A Gnome assistant (wizard/druid) to create letters (including list-merges) with support for the Gnome address book, PIM and batch operation and UNIX pipes; features user-selectable interface

modes (novice–expert) which access the features of the L^AT_EX letter class.

ifmslide (V.0.45) Produce printed slides and online presentations from only one source, add buttons for navigation and visual effects.

ifpdf.sty in *.../supported/oberdiek*
Allows detection of pdft_EX in pdf mode. Works with plain or L^AT_EX formats.

invoice.sty in *.../supported/invoice*
A tailor-made, yet expandable solution for invoicing in local and foreign currencies, including multiple assignments for a single client.

l2kurz in *info/lshort/german*
Slightly updated release of the German L^AT_EX 2_E-Kurzbeschreibung.

latex2man in *support*
V. 1.14. A tool to translate UNIX manual pages written with L^AT_EX into a format understood by the UNIX man(1)-command (or into TeXinfo or HTML).

lstpatch.sty in *.../supported/listings*
Patch to the listings package (v.0.2 1h) to fix “warning: destination with the same identifier” error with pdft_EX 3.14159-14f.

makor in *language/hebrew*
Fonts and macros to typeset Hebrew in a natural manner with a software switch to turn vowels off or on, and examples.

METAOBJ in *graphics/metapost/contrib/macros*
A large METAPOST package providing high-level objects. It implements many of PSTricks' features for node connections, but also trees, matrices, and many other things and is easily extensible.

mf2pt1 in *support*
Produces PostScript Type 1 fonts from METAFONT source in conjunction with the **t1asm** program from the **t1utils** suite.

MiniPlot Typesets eps figures using an easy-to-use interface with options for multiple figures and subfigures—wrapped figures are also supported as are informational frameboxes.

SchemeWEB2 in *web/schemeweb2*
Literate programming for Lisp and Scheme. Improved version of SchemeWEB, by John D. Ramsdell.

teubner Package for philological typesetting, especially, but not only, Greek.

ushort Updated version of this class for configurable underlining (shorter or longer, multiples) is up to four times faster.

VTeX/Free in *systems/vtex*
Release 7.33 of MicroPress' T_EX for OS/2 and Linux-x86. Supports PDF security options and **xpdf** source specials.

◊ William F. Adams
75 Utley Drive, Ste. 110
Mechanicsburg, PA 17055
USA
willadams@aol.com

Highlighting in the L^AT_EX picture environment

David M. Tulett

Emphasizing text is normally accomplished in typeset text by changing the font, which in L^AT_EX is usually done by using the `\emph` command. However, a popular way for students to emphasize what is important in their textbooks is to use a highlighter. These come in various colours, but a golden shade of yellow is most common. I wanted to create this effect in L^AT_EX documents, not only for text (which is simple), but for pictures too.

For text all that is needed is the `\colorbox` command, with a colour being chosen to mimic the typical highlighter. For this I created a colour which I named *marygold* (a play on words, it's a homonym of *marigold* but named after my wife Mary) by using the following definition:

```
\definecolor{marygold}{cmyk}%
{0,0.1,0.5,0}
```

We can then produce **highlighted text** (it will appear as a shade of grey in this publication) by using:

```
\colorbox{marygold}{highlighted text}
```

So far, everything is easy, but highlighting in the *picture* environment is not so trivial. I had made a graph in which segments of some of the lines needed to be highlighted (to show the regions of highest expected profit). The graph with the highlighted line segments appears in the figure to the right. The line marked A_4 will serve as an example. The black line was created using:

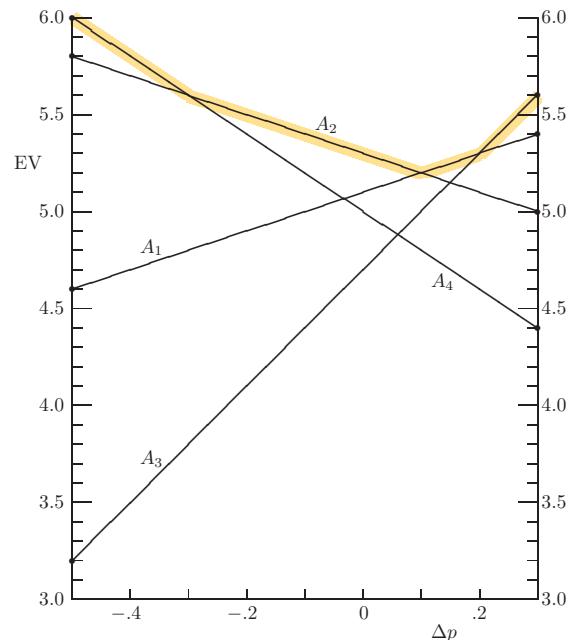
```
\put(15,350){\line(3,-2){240}}
```

This line intercepts the A_2 line at coordinate (75,310), hence the highlighter had to run from (15,350) to (75,310). To make highlighting without text, an appropriately-sized `\hspace` command is used where the text would be in the `\colorbox` command.

The `\colorbox` command needs to be embedded within a `\rotatebox` command. In conjunction with the highlighting commands for the A_2 line, there needs to be a bit of trial-and-error in positioning the commands and in specifying the length for the `\hspace` commands. The four highlighted line segments were created using:

```
% Highlighting
% A1 line
\put(194,270){\rotatebox{18.435}%
{\colorbox{marygold}{\hspace{26pt}}}}
% A2 line
\put(74,310){\rotatebox{-18.435}%
{\colorbox{marygold}{\hspace{66pt}}}}
```

```
{\colorbox{marygold}{\hspace{122pt}}}}
% A3 line
\put(222.5,279.5){\rotatebox{45}%
{\colorbox{marygold}{\hspace{37pt}}}}
% A4 line
\put(14,350){\rotatebox{-33.69}%
{\colorbox{marygold}{\hspace{66pt}}}}
```



An Example of **Highlighting** in the Picture Environment

Since the normal use of a highlighter is to superimpose it upon typeset text, one might presume that everything to appear in black would be drawn first, and that the commands to create the highlighting would come afterward. However, doing it this way creates the highlighting with the black lines underneath being removed. Instead, I discovered by experimentation that the commands to perform the highlighting need to come immediately after the `\begin{picture}` command. When I discovered this, I had incorrectly presumed that there was a special effect caused by the ordering of the commands within the *picture* environment. I am grateful to an anonymous reviewer for pointing out that this effect is caused instead by the “PostScript/PDF rendering model, which places later elements over previous elements [thereby] obscuring them.”

◇ David M. Tulett
 Faculty of Business Administration
 Memorial University of
 Newfoundland
 St. John’s, NF, Canada, A1B 3X5
 dtulett@mun.ca

Macros

A complement to \smash, \llap, and \rlap

Alexander R. Perlis

Abstract

In both plain TeX and L^AT_EX, most local alignment issues are addressed using \smash, \phantom, \vphantom, \hphantom, \llap, and \rlap. (L^AT_EX also provides \makebox. All these macros are reviewed in this article.) However, conspicuously missing is a horizontal version of \smash, which is necessary, for example, to eliminate the excessive whitespace surrounding the large operator in

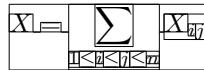
$$X = \sum_{1 \leq i \leq j \leq n} X_{ij}.$$

Another snag: whereas \smash and \phantom behave as expected in both horizontal mode and math mode, \llap, \rlap, and \makebox are not suited for use in math mode.

This article introduces the macro \clap (simultaneously a centered version of \llap/\rlap and a horizontal version of \smash), and the three macros \mathllap, \mathrlap, and \mathclap (versions of \llap, \rlap, and \clap designed for math mode).

1 Think in terms of boxes

To understand how alignment works, we should follow a TeX guru's mantra: *think in terms of boxes*.



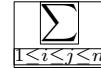
If we ignore some of the details, we are left with:

$$X = \sum_{1 \leq i \leq j \leq n} X_{ij}$$

Evidently every item typeset by TeX has two components: the ink component, and the box component. The latter is often called the “boundary box” of the item, but this can be misleading, as the box may differ radically from the tightest one surrounding the ink. *TeX’s alignment calculations are performed entirely in terms of box components*. In fact, TeX understands nothing about the ink component (such as its shape), and merely passes it along to the output file.

For individual glyphs, the box is part of the font’s design and is encoded in the font’s TFM file. That box may admit a margin or allow ink to spill out: \sum , \sqcap . But when TeX constructs a big box from many small boxes, the new outer box will be

the tightest one around all inner boxes:¹



Finally, TeX composes a line by placing boxes side-by-side without overlap.

Thus, to eliminate the excessive whitespace surrounding the large operator, we must reduce the width of $1 \leq i \leq j \leq n$ yet preserve the ink. We might as well set the width to 0. In pictures, we want to change

$$1 \leq i \leq j \leq n$$

to

$$1 \leq i \not\leq j \leq n,$$

which is a box of width 0 (indicated by a line) with ink sticking out equally on either side. The result:

$$X = \sum_{1 \leq i \leq j \leq n} [X_{ij}]$$

(What looks like a box surrounding “ $i \leq j$ ” is actually the bottom portion of the outer box that surrounds the box of the operator and the zero-width box of the subscript.)

In the next section, we’ll review the macros, available in plain TeX and L^AT_EX, that affect alignment by altering boxes. Then we’ll introduce the new macro for achieving the effect discussed above.

2 Review of existing macros

The macro \smash boxes up its material but sets the height and depth of the box to 0. Thus it is the box itself that gets smashed, not the ink in the box. To smash *both the box and the ink*, i.e., to smash the box and eliminate the ink, use \hphantom in place of \smash. Thus \hphantom produces no ink: the horizontal phantom remaining after the smashing is an infinitely thin horizontal line segment just as wide as the original material.

align	align	—
original	\smash	\hphantom

(By the way: it may be easier to understand these macros by their effect in the context of neighboring

¹ In the example here, the extra whitespace at the bottom is due to another box (more precisely: a kern) that lies below the subscript but is not indicated in our image. This mysterious box arises from the elaborate rules TeX follows for generating boxes in math mode. As explained in *The TeXbook*, Appendix G, Rule 13a: additional whitespace below subscripts (and above superscripts) of math operators is determined by \fontdimen13 of the math extension font. Rule 13a furthermore explains how TeX avoids an underfull box: it starts with the two boxes \sum and $1 \leq i \leq j \leq n$, repackages them as \sum and $1 \leq i \leq j \leq n$, and puts them (along with some kerns) on top of each other.

material, which is shown in the two tables at the end of this article.) Now compare the above with:

```
align  |  
original \phantom \vphantom
```

Evidently `\phantom` eliminates the ink without changing the box, while `\vphantom` eliminates the ink and smashes the box horizontally. (The first letter of `\hphantom` and `\vphantom` refers not to the direction of smashing but to the shape of the result.)

To smash the box horizontally, without affecting the ink, there are `\llap` and `\rlap`. The former aligns the smashed box at the right end of the ink (so that we end up with a “left overlap”), while the latter aligns the smashed box at the left end (resulting in a “right overlap”).

```
align align| \align  
original \llap \rlap
```

Finally, plain TeX and L^AT_EX diverge as follows. Missing in plain TeX is a macro we'll define in section 4 and call `\clap`, which aligns the smashed box halfway between the left and right ends of the ink (we might call this a “centered overlap”). L^AT_EX already provides it under a different name:

```
\makebox[Opt][l]{...} behaves like \llap{...}  
\makebox[Opt][r]{...} behaves like \rlap{...}  
\makebox[Opt][c]{...} behaves like \clap{...}
```

3 Concerning math mode

Whereas `\smash` and the three `\phantom` macros work correctly both in horizontal mode and in math mode, `\llap` and `\rlap` (and the L^AT_EX-only `\makebox`) are suited only for horizontal mode. To use them in math mode, we must resort to monstrosities like

```
\rlap{\$mathsurround=0pt\scriptstyle{...}}.
```

Here `\rlap` exited math mode, so we had to:

- use \$ to get back into math mode,
- use `\mathsurround` to eliminate whitespace introduced whenever we enter math mode, and
- reintroduce whatever math style was in effect before the `\rlap`.

With `\smash` and `\phantom` such shenanigans are unnecessary (indeed errors) because those macros use `\ifmmode` to test for the current mode and use `\mathpalette` to maintain the current math style. Thus where `\smash` and `\phantom` are flexible, `\llap` and `\rlap` are efficient.² Why the dichotomy? Perhaps Knuth can explain, but the mat-

² The L^AT_EX macro `\makebox` is neither flexible (in the sense under discussion) nor efficient, but has the benefit of being consistent with the rest of L^AT_EX in its use of optional parameters.

ter is moot: plain TeX is essentially frozen, and future versions of L^AT_EX are unlikely to deviate. All we can do is introduce new macros to fill in the gaps. They will be called `\mathllap`, `\mathrlap`, and `\mathclap`.³ (For L^AT_EX consistency we might also define `\mathmakebox` as a math mode analogue of `\makebox`, but don't show the code here.)

4 The new macros

Use these macros with plain TeX or with L^AT_EX.

```
% For comparison, the existing overlap macros:  
% \def\llap#1{\hbox to Opt{\hss#1}}  
% \def\rlap#1{\hbox to Opt{\#1\hss}}  
% \def\clap#1{\hbox to Opt{\hss#1\hss}}  
  
\def\mathllap{\mathpalette\mathllapinternal}  
\def\mathrlap{\mathpalette\mathrlapinternal}  
\def\mathclap{\mathpalette\mathclapinternal}  
  
\def\mathllapinternal#1#2{  
    \llap{\$mathsurround=Opt#1{#2}}}  
\def\mathrlapinternal#1#2{  
    \rlap{\$mathsurround=Opt#1{#2}}}  
\def\mathclapinternal#1#2{  
    \clap{\$mathsurround=Opt#1{#2}}}
```

5 Applications

5.1 Large operators

Excessive whitespace may be eliminated as follows:

```
X = \sum_{\mathclap{1\leq i\leq n}} X_{ij}
```

$$X = \sum_{1 \leq i \leq j \leq n} X_{ij}.$$

5.2 Tabular alignments

Consider a complicated alignment, such as polynomial long division. Fiddling with `\ialign` yields:

```
\vcenter{\def\ministrut{\vrule height2pt  
depth2pt width0pt}\offinterlineskip\ialign{  
\$mathstrut#\&\hfil\$mathsurround=0pt#\&\cr  
&&x+{}&1\alpha\cr  
\omit&\multispan{4}\rlap{\ministrut  
    \vrule height0pt}\hrulefill\cr  
x-\alpha\cr  
&\vrule height0pt\hrulefill\cr  
&x^2-{}&\alpha x\phantom{+{}+{}}\cr  
\omit&\multispan{3}\ministrut\hrulefill\cr}
```

³ Concerning names, initially I used `\hsmash` in place of `\clap`, and had it test for math mode and maintain the current math style. Thus, whereas `\smash` is like `\phantom`, my `\hsmash` was like `\vphantom`. Concerned that the names were confusing, I pondered: clapping one's hands together might be the horizontal analogue of smashing one's hands on, say, a desk. Thus the name `\clap` was born, but the definition still mimicked `\smash`. Only later did I realize the obvious connection (both in name and behavior) with `\llap`/`\rlap`. Remembering the separate need for math versions of those macros, I arrived at the design presented in this article: three overlap macros for horizontal mode, and separately three overlap macros for math mode.

```
&&&(1+\alpha)x+{}&2\cr
&&&(1+\alpha)x-{}&\alpha-\alpha^2\cr
\omit&&&\multispan{2}\ministrut\hrulefill\cr
&&&2+\alpha+\alpha^2\cr
\}
```

$$\begin{array}{r|cc} & x+ & 1+\alpha \\ x-\alpha & \begin{array}{c} x^2+ \\ x^2- \end{array} & \begin{array}{c} x+ \\ \alpha x \end{array} \\ \hline & \begin{array}{c} (1+\alpha)x+ \\ (1+\alpha)x- \end{array} & \begin{array}{c} 2 \\ \alpha-\alpha^2 \end{array} \\ & & 2+\alpha+\alpha^2 \end{array}$$

By inserting `\mathllap` thrice, `\mathrlap` twice, and `\quad` once (exercise: determine where), we reduce whitespace and allow α^2 to stick out:

$$\begin{array}{r|cc} & x+1+\alpha \\ x-\alpha & \begin{array}{c} x^2+ \\ x^2- \end{array} & \begin{array}{c} x+ \\ \alpha x \end{array} \\ \hline & \begin{array}{c} (1+\alpha)x+ \\ (1+\alpha)x- \end{array} & \begin{array}{c} 2 \\ \alpha-\alpha^2 \end{array} \\ & & 2+\alpha+\alpha^2 \end{array}$$

5.3 Commutative diagrams

Consider the alignment of arrows, objects, and arrow labels in commutative diagrams. Because many diagram packages exist, instead of showing the source for the following simple diagrams, the onus is on the reader to reproduce the following effect using the diagram package of choice.

By putting all the primes inside `\mathrlap`,

$$\begin{array}{ccc} C \xrightarrow{\phi} C' & & C \xrightarrow{\phi} C' \\ \downarrow & \downarrow & \downarrow \\ C'' \xrightarrow{\phi''} C''' & \text{might become} & C'' \xrightarrow{\phi''} C''' \end{array}$$

I wrote *might* because some minor additional fiddling may be necessary due to the difficulty that the box of an entry such as `C\mathrlap{''}` surrounds only the “C”; consequently, the neighboring arrow may land on top of the primes. Depending on your choice of diagram package, you might work around this problem by demanding some entries to have a wider margin, or by defining a new horizontal arrow that has some extra space at one end. (The diagrams above were produced using `\ialign`, so I simply preceded each arrow with `\mskip\thinspace`.)

6 Conclusion and acknowledgment

The new macros complement the existing ones by filling in the obvious gaps, as is evident from the tables below. My hope is that these macros (along with `\mathmakebox`—see section 3) will be incorporated into a future version of L^AT_EX, or at least become part of the `amsmath` package of *AMS-L^AT_EX*.

MACRO	MODE	EFFECT	IN CONTEXT
none	either	<code>align</code>	
<code>\smash</code>	either	<code>align</code>	
<code>\phantom</code>	either		
<code>\hphantom</code>	either		
<code>\vphantom</code>	either		
<code>\llap</code>	horiz.	<code>align </code>	
<code>\rlap</code>	horiz.	<code>align </code>	
<code>\clap</code>	horiz.	<code>align </code>	
<code>\mathllap</code>	math	<code>align </code>	
<code>\mathrlap</code>	math	<code>align </code>	
<code>\mathclap</code>	math	<code>align </code>	

Table 1: Effect of existing macros (first six) and new macros (last four).

COMBINATION	MODE	IN CONTEXT
<code>\llap{\smash{...}}</code>	horiz.	<code>align</code>
<code>\rlap{\smash{...}}</code>	horiz.	
<code>\clap{\smash{...}}</code>	horiz.	
<code>\mathllap{\smash{...}}</code>	math	<code>align</code>
<code>\mathrlap{\smash{...}}</code>	math	
<code>\mathclap{\smash{...}}</code>	math	

Table 2: The remaining effects are achieved using combinations.

For useful comments on an earlier version of this article, and in particular for telling me about `\makebox`, I thank Michael Downes.

◊ Alexander R. Perlin
 Department of Mathematics
 The University of Arizona
 Tucson, AZ 85721 USA
 aprl@math.arizona.edu



Typesetting critical editions of poetry

John Burt

Abstract

`poemscol` provides macros for LATEX for setting collections of poetry. It is especially suited for setting collections of poetry in which several volumes are combined, such as in a critical edition of a poet's Collected Poems. It provides the structures required to produce a critical edition of the kind specified by the Modern Language Association's Committee on Scholarly Editions, such as line numbering of the poems and multiple series of endnotes tied to the line numbers of the poems, and it automatically marks every occasion where a stanza break falls on a page break. It provides running headers of the form "Notes to pp. xx–yy" for the notes sections, and other structures such as the table of contents, the index of first lines and titles, divider pages that separate sections of the book, and automatic adjustment of the running headers for the different sections of a Collected Poems volume.

1 Critical editions

1.1 What a critical edition is

A critical edition is a special version of a literary work in which the editor has, by collating the manuscripts, typescripts, galleys, or published versions of a text, and by examining other kinds of relevant evidence (such as the author's correspondence with publishers), attempted to produce a text that accurately represents what the author intended to write. While scholars continue to disagree about what constitutes authorial intentions, about what counts as evidence for authorial intentions, and even about whether the concept of authorial intention is ultimately a meaningful one, critical editions remain essential tools for the study of literary works, making literary works available for the next generation, and giving reasoned accounts of the editor's views of what the author did or did not actually say.

Every scholarly editor has heard the story about how F. O. Matthiessen, in his pathbreaking book *American Renaissance*, provided an illuminating commentary upon a passage in Melville's *White-Jacket* in which Melville spoke of a sailor falling from a masthead into the ocean, where he brushes a "soiled fish of the sea." Matthiessen brooded at length upon how a fish could be soiled, unaware that

the line is the work of the compositor of the reprint edition he had been reading; in the first edition, it is a coiled fish of the sea that the sailor brushes. Without critical editions, literary critics wind up catching all too many "soiled fish of the sea."

Critical editing, of course, has a long history, stretching back to Renaissance humanists' attempts to determine an accurate text of the Bible, but it assumed its modern form in the middle of the last century, as W. W. Greg, Charlton Hinman and others developed methods to establish the texts of Shakespeare's works. Thomas Tanselle's *A Rationale of Textual Criticism* provides a thorough, if conservative, treatment of the history and method of the scholarly editing of critical editions.

The importance of critical editions is more than merely scholarly. Only since the appearance in 1955 of Thomas Johnson's edition of the poetry of Emily Dickinson (recently superseded by Ralph Franklin's new edition) have readers been able to see how unconventional her poetry is, since only with Johnson's edition were readers finally able to separate what Dickinson wrote from the various ways her early editors altered the texts of her poems to suit their own literary sensibilities, regularizing Dickinson's meters and straightening out her slanted rhymes. Critical editions of historical documents are also important, but the canons of editing them are slightly different from those employed for literary texts.

1.2 Features of critical editions

Critical editions have a number of features in common, many of them specified by the Committee on Scholarly Editions of the Modern Language Association. These include such things as an essay describing and defending the editor's choice of copy text (ideally, say, the fair copy of the manuscript that the author sent to the publisher) and an account of the rules the editor employed in emending it, an introduction laying out some of the key literary issues about the text, an index of titles and first lines, and several varieties of notes.

These notes, which may be footnotes or endnotes, typically include several series. Minimally they include a list of emendations, accounting for every instance in which the editor has chosen to depart from the copy text, and a set of explanatory notes, making clear allusions or historical references or other kinds of background information the editor feels to be necessary in order to understand particular passages.

Critical editions also usually include textual collations, sometimes distinguishing what are called

“substantive” variants (which are, roughly speaking, variants in the actual words in the text) and what are called “accidental” variants (which are, roughly speaking, variants in the punctuation of the text). The rule of thumb about substantive and accidental variants is that if a substantive variant appears in a late published version for which the author read proof, it is highly likely to be a considered revision of the text by the author, but an accidental variant in a late version for which the author read proof is less likely to be authorial, since authors are less likely, when reading proof, to notice accidental variants than substantive ones.

In critical editions of prose works, it is the general practice to include an account of hyphenations at the ends of lines, distinguishing between instances in which the hyphen is simply an artifact of a line-ending (whether in the copy text or in one’s own edition) and instances in which the hyphen should be preserved when quoting the text.

1.3 Other critical edition packages

Critical editions are time-consuming to prepare and expensive to publish, and *T_EX* is ideally suited for automating some of the most tedious (and error-prone) tasks, for lowering the cost of producing a published edition, and for removing the publisher’s typesetter, a possible source of new errors, from the production of the document. There are several packages available for critical edition typesetting on CTAN. Of these, the gold standard is undoubtedly **EDMAC**, a plain *T_EX* format by John Lavagnino and Dominik Wujastyk. **EDMAC** is tremendously flexible and tremendously feature-rich; in addition, it has a long history of actual use among publishers of critical editions. Most of **EDMAC**’s features have been ported to the *L^AT_EX* world with **ledmac**, a *L^AT_EX* style package by Peter Wilson. A somewhat different approach has been taken by Uwe Lück in the **ednotes** package, which combines existing *L^AT_EX* macro packages such as **lineno** and **manyfoot** to meet the needs of critical editions.

2 Special markup for poetry

2.1 Verse lines and physical lines

poemscol differs from these other packages in being specifically designed for verse. At first glance, this means only that it does not use the sophisticated paragraph decomposition technique that **EDMAC** uses in order to attach line numbers to prose paragraphs. But it also means that **poemscol** pays attention to the key feature that distinguishes prose from verse, which is that a verse line is a unit of versification,

not a unit of typesetting, and may run over several physical lines (if the poet writes long lines, as Whitman did), or may be broken over several physical lines (if the poet, say, shifts speakers or subjects in mid-line). The verse line also has a complicated relationship to the stanza or verse paragraph, with some poets breaking a single line across a stanza break, as Robert Penn Warren often does.

poemscol is designed for typesetting lyric and narrative poetry, but it has provisions for poems that interrupt their verse with prose interludes (**poemscol** will not line-number the prose passages, however, although it will remember what the line number was when the verse picks up again). With a certain amount of fiddling, **poemscol** could be modified to typeset dramatic verse, although it doesn’t provide markup for the special structures for drama (stage directions, speech tags, act and scene environments, and so on) right now. There are, however, *L^AT_EX* style files and class files with all of the structures for typesetting drama available on CTAN, and there is no reason to assume that **poemscol** could not work in conjunction with them.

2.2 Marking Logical Units of Poems

With **poemscol** you mark poem titles, lines, stanzas, textual notes, emendations, explanatory notes, and entire poems up as logical units, and *L^AT_EX* does your formatting, and your counting, for you. Once you have marked out the logical units of the poem, **poemscol** will take care of the line numbering, and will automatically mark every textual note, emendation, or explanatory note with the line number.

One advantage of this kind of markup is that even if the appearance of the poem on the page may be ambiguous, the editor’s intentions about the logical structure of the poem will be preserved in the *L^AT_EX* source. **poemscol** is designed so that even readers who know no *L^AT_EX* can understand the editor’s intentions by examining the markup. By manipulating penalties and designing commands even for relatively unusual circumstances, **poemscol** attempts to make it possible for your source files to contain markup that is almost entirely content-based, marking the nature of the poetic objects to be set, with only a little markup of an explicitly typographical character. Although *L^AT_EX* is a typesetting language, not a content markup language, *L^AT_EX* comes very close to enabling one to realize the ideal of completely separating content markup from formatting, and I have tried to design **poemscol** with that ideal in mind.

Should you wish later to produce an electronic edition of your work, either using XML or the SGML

markup approved by the Text Encoding Initiative, transforming your texts from L^AT_EX to XML would largely (although not entirely) be a matter of performing a series of global search-and-replaces, and could conceivably be done with a perl script.

2.3 Stanza breaks at page turns

In poetry which does not have a regular stanzaic form, it is useful to be able to mark occasions where there is a stanza break at the bottom of a page which the reader might not notice. When poets submit their typescripts to publishers, they often painstakingly mark every occasion in which a stanza break falls on a page turn, since the typesetter might omit the stanza break otherwise. But the publishers themselves mostly do not mark stanza breaks at page turns, leaving it ambiguous for the reader whether there is a stanza break at that point or not. All publishers of poetry should mark stanza breaks at page turns, but few do. Marking such stanza breaks by hand is not only tedious and easy to get wrong, but also a process you will have to start over again if anything about your volume changes, if you add a poem, say, or even decide to break a title across two lines. `poemscol` automatically marks cases where the page break coincides with a stanza break by placing a symbol in the running footer. To do this, `poemscol` changes the value of `\mark` when one is inside a poem environment, but not inside a stanza environment. You can decide to mark cases where there is a stanza break at a page turn, or cases where there is not a stanza break at a page turn, or both, and you can choose what mark to put in the running footer for both cases.

2.4 Titles of poems and poetic sequences

The command names may seem ugly and long. And there are separate commands for many tasks that seem closely related, such as a command to mark the title of a section of a poetic sequence, and a separate command to mark a subsection. But the names do describe pretty much what each macro does, and they do specify exactly what the object they mark is supposed to be.

There are also many commands for special purposes whose necessity may not seem clear until the editor finds him or herself in the jam the command was designed for. If, for instance, a long poem has several sections whose titles are only roman numerals, the ordinary `\poemsectiontitle{}` command may be inadequate, since there may be other poems in sections that do the same thing, and `\poemsectiontitle{}` will confuse the cross-references `poemscol` uses to set the page numbers

that refer to the section in the textual notes. The `\poemsectiontitlenocontents{}{}` macro solves this problem by having two arguments, the first for the section title, and the second for the poem title, so that every section simply titled “III” will have a unique cross reference.

`\poemtitle{}` (and a host of related commands such as `\sequencetitle{}` and `\sequencesectiontitle{}`) sets the title of the poem. `poemscol` gives default values for such things as the font size, the separation between the top of the title and the bottom of the previous poem, the separation between the bottom of the title and the first line, and so on. It also sets penalties in order to encourage page breaks just before a title, and to discourage page breaks between a title and a poem. Finally, it sends information about the poem to the external files that manage the table of contents and the various notes sections about the poem.

Tables of contents of collections of poetry are unlike tables of contents of scholarly books, and for that reason I have written macros for making such a table of contents to use in place of the macros from L^AT_EX. These macros are rather primitive, and lack many of the better features of tables of contents in L^AT_EX, but they do give a consistent look, and some thought has been given to distinguishing volumes, poems, poetic sequences, and sections of poetic sequences in the table of contents.

Every `\poemtitle{}` will also open a new paragraph entry in the “Textual Notes” section (if compiling textual notes is enabled) headed with the page number and title of the poem, since presumably one will want to give information about the publication history of every poem. The `\poemtitle{}` command and its fellows, however, only open new paragraph entries in the “Explanatory Notes” and “Emendations” sections (if compiling these is enabled) if there actually are explanatory notes and emendations for that poem in the text that follows.

There are special commands for titles with multiple lines, for titles with italicized words or other formatting, and for poems without formal titles (which may be listed in the table of contents and in the notes sections by their first lines). There is even a special command for poems without formal titles whose first lines include italicized words. By default, the titles of poems with multiple line titles will be broken the same way in the table of contents that they were broken in the body of the text, but this can be changed by using the `nocontents` or `baretitle` forms of the title commands together with `\literalcontents{}`. In the notes sections, the

titles of poems with multi-line titles will be run in by default, although you could change this using the `nonotes` or `baretitle` forms of the title commands together with `\literaltextnote{}` and its siblings.

Sequences of related poems, perhaps with an over-title, demand special handling in the text, in the notes sections, and in the table of contents. (A moment's thought will show that they are different in some ways from poems in sections. For one thing, a sequence manifests a different relationship between part and whole than a poem in sections does, and often provides stronger experiences of momentary closure between sections.) `\sequencefirstsectiontitle{}` gives a little more vertical space between the main title and the first section title than `\sequencesectiontitle{}` gives between adjacent sections later on in the sequence (since the over-title for the sequence is in larger type) and adjusts the page breaking penalties to reflect the fact that such titles should not occur near the bottom of a page, since there should be no page break between the sequence title and the title of the first section of the sequence (although page breaks are permissible, indeed even slightly favored, between later sections). There should also never be a page break between a section title and the first or second line of the section. The penalties `poemscol` sets should take care of these automatically, but you may still have cases which require you to specify a page break with `\pagebreak` or `\newpage`. There are a host of other commands, all with self-explanatory titles, that deal with some special situations that arise in setting the elements of poetic sequences. These macros don't of course exhaust the dizzying possibilities, but from them you can construct whatever other macros you may need. (For long poems in books or cantos, it would be wise to use `\poemtitle` for the canto names or numbers, setting the volume title in some other way, perhaps with `\volumetitle` described above.)

Finally, the elegantly named commands `\poemsubtitle{}`, `\epigraph{}`, `\dedication{}`, and `\attribution{}` are for, well, subtitles, epigraphs, dedications and attributions.

2.5 The poem environment

The body of every poem should be placed in a poem environment. Putting the body of the poem between `\begin{poem}` and `\end{poem}` resets the line counter to 1, and puts the poem in a `\verse` environment (to handle run over lines automatically). `poemscol` slightly modifies the `\verse` environment from the standard L^AT_EX definition, increasing the indentation used for run over lines, in order to make

the difference between the indented run over portion of a long line, on one hand, and an explicitly indented second line, more obvious in the output.

2.6 No hyphenation in verse

`poemscol` turns off automatic hyphenation in poetry environments. The idea here is that every hyphen in the printed poem is authorial, obviating the need for you to compile a hyphenated-lines list to distinguish between authorial hyphens and hyphens added for lineation purposes. Since poetry is normally not set with a flush right margin, giving up hyphenation is no hardship. (You may wish to change this for your own edition, but if you do so you must keep track of added hyphens yourself. This list will be easy to compile, however, because only authorial hyphens will appear in your source code. Automatically added hyphens will appear only in the output. You might even modify the output routine so that automatically added hyphens have a different look.)

`poemscol` turns automatic hyphenation back on in prose contexts, so if you wish to keep a hyphenation list for such things as authorial prefaces and so on, you must do so yourself manually. (Alternatively, you can turn automatic hyphenation off in those contexts as well, by setting the `\language` to 255. If you do turn automatic hyphenation off, it would be wise to restrict the change to some particular environment, rather than changing the `\language` globally.)

2.7 The stanza environment

Every stanza should be placed in its own stanza environment. Every poem should have at least one stanza. Marking the beginning and end of every stanza (with `\begin{stanza}` and `\end{stanza}`) provides `poemscol` with a way of detecting cases in which a page boundary falls on a stanza break, since in those cases a page turn happens when one is *inside* a poem environment but *not* inside a stanza environment. Further, marking the beginning and end of every stanza makes the logical structure of the poem (and the editor's intentions about it) clear to readers of your source code. `poemscol` adds a little bit more white space between stanzas than the standard L^AT_EX `verse` environment does. (I found that the standard stanza breaks did not leap out on the page as stanza breaks.) `\verseline` should mark the end of every line, except the last line of every stanza (which should be marked with `\end{stanza}`).

2.8 Broken lines of various kinds

`poemscol` automatically runs over long lines, indenting the run over portion on the next physical line. If you are unhappy with where `poemscol` has run over a particular line, you can “bend” that line by issuing `\linebend` at the point where you wish it to run over. The run over portion of the line will be indented just as if `poemscol` had “bent” the line at your selected point. This command only works if you have chosen to bend the line at some point earlier than `poemscol` would have chosen on its own. If you really do want to extend a line further into the right margin, you can probably do so by using a combination of `\nobreak` and `\hbox{}`, or by turning all of the spaces in that line into unbreakable spaces, marked with `~` in your source. But L^AT_EX will complain if you do this, and rightly so, since the result is likely to be ugly. You may also wish to use `\linebend` to reproduce how your author broke up long lines on the page (if you know that your author cared about such things and did not leave them up to the typesetter).

`\linebend` should only be used for managing run over lines, not for cases in which a line is to be broken into separate half-lines. For cases in which a line is to be broken into half-lines, use the `\brokenline` macro. The two macros do similar (but not identical) things. But a “linebend” is a feature of typesetting, and a “broken line” is a feature of versification, and it seems best to distinguish them logically. (`\linebend`, like `\brokenline`, issues a carriage return without incrementing the line number, but `\linebend` adds indentation to the next line.)

`\brokenline` is normally used with `\versephantom{}`, which adds white space exactly as long as its argument would have been had it been set in type. `\versephantom{}` thus provides an easy way of setting the beginning of the second half-line flush with the end of the first, whatever the font size or special formatting of the first line.

The sestet of Yeats’s sonnet “Leda and the Swan,” has such a broken line:

A shudder in the loins engenders there
The broken wall, the burning roof and tower
And Agamemnon dead.

Being so caught up,
So mastered by the brute blood of the air,
Did she put on his knowledge with his power
Before the indifferent beak could let her drop?

To set the broken line properly, issue:

```
And Agamemnon dead.\brokenline  
\versephantom{And Agamemnon dead.}  
Being so caught up,\verseline
```

Some poets occasionally introduce a stanza break in the middle of a broken line, considering the line to be a single metrical unit despite the fact that it straddles a stanza break. To record these cases, mark the end of the first half-line with `\end{stanza}` as usual. But instead of opening the next stanza with `\begin{stanza}` issue `\stanzalinestraddle` instead. This will make sure that the line counter counts the straddling line as only one line, despite the stanza break. `\stanzalinestraddle` is usually used with `\versephantom{}`.

3 Collations, emendations, and explanatory notes

3.1 Initialization of endnote sections

`poemscol` makes textual notes of various kinds. It can set textual collations as footnotes, but it is designed to set collations, emendations, and explanatory notes in separate endnote sections as block paragraphs headed with the page number and title of the poem they concern. `poemscol` will automatically generate a running header of the form “Emendations to pp. xx–yy” for the notes sections. (To do this, `poemscol` uses the `\mark` mechanism. Every time `poemscol` writes a note to one of the external files for notes, it also writes out a string which, when read back in, is a little program which records the page number of the main text that was being set when the note was written out to the external file, compares it to the page number that was written out for the first note on that page in the notes section, and adjusts the `\mark` for the running header accordingly.) To collect these textual notes, issue `\maketextnotes`, `\makexplanatorynotes`, and `\makeemendations` in your preamble. The notes sections, and the table of contents section, write external files with characteristic extensions. (The code for doing this is borrowed from John Lavagnino’s `endnotes` package.) To set these sections in the proper place, you will either need to `\input` them in your driver file, or use the `\finish` macro.

3.2 Recording notes

To record information about the copy text, editions and publication history of individual poems, or any information not tied to specific lines in the poem, you should place that information in the argument to the `\sources{}` macro. Typically, you should issue this macro after you have issued `\poemtitle{}` and before you issue `\begin{poem}`. If you wish to send information to the textual notes file (such as to force a page

break), you can do so by using `\sources{}`. You can send typesetting information to other sections by using `\literalemend{}`, `\literalexplain{}`, or `\literalcontents{}`. There is also a `\literaltextnote{}` command, which is equivalent to `\sources{}`.

`\textnote{}` is used to capture variants and tie them to the correct line number. Issue `\textnote{}` immediately after the `\verseline` command which marks the ending of the line you wish to comment upon. Put the text of your note (which may be simply the recording of a variant in the standard notation) into the argument of the macro. You should put both the lemma and the variants or comment in the argument to the `\textnote{}` macro.

To put the \sim glyph in your note (used for recording places where the variant and the copy text have the same word, as for instance when recording a variation of punctuation) use `\sameword`. To put the \wedge glyph into your text (used for recording places where a punctuation mark is missing in a variant), use `\missingpunct`.

`\emendation{}` and `\explanatory{}` are used exactly as `\textnote{}` is. Issue the emendation or the explanatory note as the argument to the command. Place the command immediately after the `\verseline` that concludes the line upon which it is a comment.

3.3 Accidentals and typescript variants

`\accidental{}` behaves exactly like `\textnote{}`. If you wish to distinguish between accidentals and substantives, this provides a way of doing so. If you wish to include these accidentals in your textual collations, issue `\global\includeaccidentalstrue` in your preamble.

`poemscol` does not provide for a separate back-matter section for accidentals, but it would be trivial to construct one, creating a `\makeaccidentals` command on the analogy with `\maketextnotes` and redefining the `\accidental{}` macro to divert its output into a new, separate external file.

If you wish to exclude accidentals from your printed output, but to mark them in your source files, so that your published collation consists only of substantives, issue `\global\includeaccidentalsfalse` in your preamble. Many publishers are reluctant to publish accidentals, believing that they are, well, less substantive than substantives. Using the `\accidental{}` command allows you to exclude accidentals from the published version should your publisher insist, while preserving the information about them should the publisher's mind change.

In the very worst case, if you have marked all the accidentals in this way you can still produce a list of accidentals for later use, and other scholars can search for accidentals in your source files simply by searching for the string `\accidental`.

If you wish to distinguish between published variants and typescript, manuscript, or galley variants, `\tsvariant{}` provides a way of doing so. If you wish to include these variants in your textual collations, issue `\global\includetypescriptstrue` in your preamble, in which case `\tsvariant{}` will behave exactly like `\textnote{}`. To exclude typescript variants, issue `\global\includetypescriptsfalse` in your preamble. (Some publishers may turn up their noses at typescript variants in just the way they turn up their noses at accidentals.)

If you wish to include typescript entries in a single note including those entries in a list with variants from other published versions (as for instance when a comma appears in a typescript but only in the second edition of the published poem), simply issue `\textnote{}` as usual, marking the relevant variant in the list of variants with the `\tsentry{}` macro. If `\global\includetypescriptstrue` appears in your preamble, the entry will be included in that textual note. If typescript variants are excluded, the typescript entry will also be excluded. You can mark individual variants with `\tsentry{}` in the arguments to the `\explanatory{}` and `\emendation{}` commands as well.

Here is a typical use of the `\tsentry{}` command:

```
[\small]
Of moonlit desert. A stallion, white and
flashing, slips,\verseline
\textnote{Of moonlit} Of the moonlit
{\em NY/}\tsentry{, SP85TS
(revised in black pen to SP85)}}
```

In the example, the version of the poem published *The New Yorker* includes a variant, and the variant is shared with the typescript for the 1985 *Selected Poems*, but crossed out on the typescript, and not included in the published version of SP85. If `\includetypescripts` is set to false, then the note in the output will show the variant from *The New Yorker* but not the variant from SP85TS. Notice that since the `\tsentry{}` comes in the middle of the list, it begins with a comma, and there is no white space between that `\tsentry{}` and the previous entry.

4 Other structures

Editions of collected poetry might also require special structures to reflect the fact that they are made up of the contents of several volumes of poetry. In particular, such editions require special structures for setting up specially formatted divider pages between volumes. They also require tables of contents and other front matter. `poemscol` provides these structures.

4.1 The main title page and divider pages

The `\volumetitlepage` environment is an environment for divider pages in collections made up of several volumes. Volume title pages will always appear on recto pages, and the following verso page will be empty. `poemscol` will also automatically create a blank verso page preceding the volume title page if necessary. `\volumeepigraph{}}, \volumeattribution{}}, and \volumededication{}` mark out epigraphs, attributions of epigraphs, and dedications on divider pages or on the main title page. The `\maintitlepage` environment is for the title page of the whole book. The main title page will also automatically always be on a recto page, with an empty verso. These divider pages, and their blank versos, have special page styles, with no page numbers and no running headers.

4.2 Table of contents

`poemscol` will also create a table of contents. To make a table of contents, issue `\makepoemcontents`. Generating the information for the table of contents will take three passes; one to generate the cross-reference information for the titles, one after running `MakeIndex` to include the index of titles and first lines in the table of contents, and one to set the contents. You will need to `\input` the contents file (with extension `.ctn`) in the proper place, commenting the line out in your driver file until the final run.

4.3 Index of titles and first lines

The Index of titles and first lines is generated by `MakeIndex` in the usual way, but `poemscol` provides formatting information for that index, and automatically provides for adding an entry about the index to the Table of Contents.

5 Example

Figure 1 shows how a poem with a complex publication history might be marked up with `poemscol`, using the `poemscol` source for a poem Robert Penn Warren wrote in his first volume, *Thirty-Six Poems*

(1935). The poem is the first poem of his sequence “Kentucky Mountain Farm.” It was included in my edition of *The Collected Poems of Robert Penn Warren*, which was set using `poemscol`. For purposes of example, I include the over-title, all necessary package inclusions and other boilerplate (see the `poemscol` package documentation for more details). Figures 2–5 show the corresponding output.

6 Conclusion

Critical editions with all the trimmings, such as the volumes of Herman Melville produced by the Northwestern University / Newberry Library group, or the Thomas More project at Yale, are ambitious undertakings which require large resources and large measures of patience, and publication of such editions is itself highly expensive. But the production of critical editions plays a crucial role not only in giving shape to the oeuvre of established authors, but also in broadening the literary canon to include authors hitherto neglected. The development of inexpensive software, such as the several packages for critical editions found in the `TEX` world, can, it is hoped, make critical editions less of a niche product.

References

- Burt, John, editor. *The Collected Poems of Robert Penn Warren*. Louisiana State University Press, Baton Rouge, 1998.
- Franklin, Ralph W., editor. *The Poems of Emily Dickinson*. Harvard University Press, Cambridge, MA, 1998.
- Greg, Walter Wilson. “The Rationale of Copy-Text”. *Studies in Bibliography* 3, 19–37, 1950–1.
- Greg, Walter Wilson. *The Editorial Problem in Shakespeare; a Survey of the Foundations of the text*. Oxford University Press, Oxford, 1951.
- Hinman, Charlton. *The Printing and Proof-reading of the First Folio of Shakespeare*. Oxford University Press, Oxford, 1963.
- Johnson, Thomas H., editor. *The Complete Poems of Emily Dickinson*. Little, Brown, Boston, 1955.
- Matthiessen, F. O. *American Renaissance; Art and Expression in the Age of Emerson and Whitman*. Oxford University Press, New York, 1941.
- Tanselle, G. Thomas. *A Rationale of Textual Criticism*. University of Pennsylvania Press, Philadelphia, 1989.

John Burt
Department of English MS023
Brandeis University
Waltham, MA 02454

```
\documentclass[10pt,twoside]{article}
\usepackage{fancyhdr,makeidx,multicol}
\usepackage{keyval,ifthen,newmarn}
\usepackage{geometry,poemscol}
\begin{document}
\pagestyle{empty}
\setcounter{page}{35} % for our example
\leftheader{The Collected Poems of
    Robert Penn Warren}
\makeexplanatorynotes
\makeemendations
\maketextnotes
\makepoemcontents
\makelinenumbers
\global\indexingontrue
\global\includeaccidentalstrue
\global\includetypescriptstrue
\sequencetitle{Kentucky Mountain Farm}
\index{Kentucky Mountain Farm@{\em Kentucky Mountain Farm\!}}
\sources{Text: TSP. Variants:
SP43, SP66 (Deletes "The Cardinal,"'
"‘The Jay,’" and "‘Watershed’"), SP75
(Same sections as SP66), SP85 (Restores
"‘Watershed’"), {\em Helsinki\!}
[...]
}
\sequencefirstsectiontitle{I. Rebuke
of the Rocks}\index{Rebuke
of the Rocks @{\em Rebuke of the Rocks\!}}
\sources{Text: TSP.
Variants: {\em Nation\!}, 11 Jan.\ 1928,
p.^47, {\em Literary Digest,\!}
28 Jan.\ 1928, p.^32, {\em Vanderbilt
Masquerader,\!} 10 (Dec.\ 1933), p.^16,
SP43, SP66, SP75, SP85, {\em Helsinki\!},
[...]
}
\begin{poem}
\begin{stanza}
Now on you is the
hungry equinox,\verseline
\index{Now on you is the hungry equinox}
O little stubborn people of
the hill,\verseline
\accidental{hill,} \sameword---
{\em Nation,\!}
{\em Literary Digest\!}
\sameword, {\em Vanderbilt\!} (I include
[...]
ironwood.\end{stanza}
[...]\end{stanza}
\end{poem}
\finish % comment out on first run
\end{document}
```

Figure 1: poemscol input example for *Kentucky Mountain Farm*.

Kentucky Mountain Farm

I. Rebuke of the Rocks

Now on you is the hungry equinox,
 O little stubborn people of the hill,
 The season of the obscene moon whose pull
 Disturbs the sod, the rabbit, the lank fox,
 Moving the waters, the boar's dull blood,
 And the acrid sap of the ironwood.

But breed no tender thing among the rocks.
 Rocks are too old under the mad moon,
 Renouncing passion by the strength that locks
 10 The eternal agony of fire in stone.

Then quit yourselves as stone and cease
 To break the weary stubble-field for seed;
 Let not the naked cattle bear increase,
 Let barley wither and the bright milkweed.
 Instruct the heart, lean men, of a rocky place
 That even the little flesh and fevered bone
 May keep the sweet sterility of stone.

Figure 2: Output of *Kentucky Mountain Farm*.

CONTENTS	
Kentucky Mountain Farm	
	I. Rebuke of the Rocks / 35
	Emendations / 37
	Textual Notes / 39
	Explanatory Notes / 41
	Index of Titles and First Lines / 43

Figure 3: Table of Contents.

INDEX OF TITLES AND FIRST LINES	
<i>Kentucky Mountain Farm</i> , 35	<i>Rebuke of the Rocks</i> , 35
Now on you is the hungry equinox, 35	

Figure 4: Listing of first lines.

TEXTUAL NOTES

35 **Kentucky Mountain Farm** Text: TSP. Variants: SP43, SP66 (Deletes “The Cardinal,” “The Jay,” and “Watershed”), SP75 (Same sections as SP66), SP85 (Restores “Watershed”), *Helsinki* (includes only “Rebuke of the Rocks” and “At the Hour of the Breaking of the Rocks”). “The Owl” (above) was marked as a section of “Kentucky Mountain Farm” when it first appeared in *Poetry*, but it was never included in any book version of the entire sequence. The sequence in *Poetry* included, in this order, “The Owl,” “The Cardinal,” and “Watershed.” TSP uses lower case Roman numerals in the section titles. The typescript drafts in the Beinecke Library do not seem to be setting copies.

35 **I. Rebuke of the Rocks** Text: TSP. Variants: *Nation*, 11 Jan. 1928, p. 47, *Literary Digest*, 28 Jan. 1928, p. 32, *Vanderbilt Masquerader*, 10 (Dec. 1933), p. 16, SP43, SP66, SP75, SP85, *Helsinki*, Broadside: The Press at Colorado College, printed on paper handmade by Thomas Leech for the American Poetry Society, April 26, 1985. This poem was not included in SP85 until the second set of galleys, in which a photocopy of the SP75 text is a stapled insert. 2: hill,] ~— *Nation*, *Literary Digest* ~, *Vanderbilt* (I include the reading from *Vanderbilt* even though it is the same as in TSP, because *Vanderbilt* was published after the other magazine versions but before TSP.) 8: old,] ~, *Vanderbilt* 11: stone,] ~, *Vanderbilt* 14: milkweed.] milk-weed. *Vanderbilt*

Figure 5: Textual notes. The ‘35’ which begins each paragraph refers to the page number where the poem appears.

CV formatting with **C_urV_e**

Didier Verna

Abstract

C_urV_e is a L^AT_EX 2_< class package for writing curricula vitae. It provides a set of commands to create headers, rubrics, entries in these rubrics and so on. **C_urV_e** will then format your CV with a consistent layout, while you can just concentrate on the contents. **C_urV_e** has a very special feature known as the *flavor mechanism*: it is able to manage different “flavors” (versions) of your CV simultaneously.

C_urV_e is distributed under the terms of the LPPL license. This paper gives an overview of the features available in version 1.4.

— * —

1 Yet another CV class?

The first draft of **C_urV_e** appeared in 2000, when I was completing my Ph.D. and was starting to look for

a job. At that time, several options were available: using a WYSIWYG editor, using plain (L^A)T_EX (that is, with no particular class), or using an already existing CV class for L^AT_EX. After some investigation, none of these options turned out to be satisfactory for me.

As the visual aspect of a CV is something very important, one could be tempted to use a WYSIWYG editor, thinking that the layout conception would be faster. For me, this was (and still is) wrong:

- Firstly, WYSIWYG editors are usually far from it: what you see (on screen) is most of the time *not* what you get (on paper). Hence, WYSIWYG editors often turn out to be a handicap rather than a help.
- Secondly, a good CV requires a very strict layout, of the kind on which you spend hours, unless it is completely automated. Doing this with a WYSIWYG editor would involve a deep knowledge of the editor itself (for instance, knowing

Didier Verna
Born April 21st 1970
French

<mailto:didier@lrde.epita.fr>
<http://www.lrde.epita.fr/~didier>

Curriculum Vitae

Sample made with CurVe

Professional Experience

Lecturing

- **L^AT_EX 2_ε**: an overview. 3 hours conference.
- **OpenGL Programming**. 15 hours lecture, including workshops.
- **Operating Systems**. 30 hours lecture.

Development

- L^AT_EX 2_ε
XEmacs
GNU
- Author of CurVe, FiNK and FiXme.
 - Member of the review board.
 - Contributor to other free software projects.

Education

- | | |
|-----------|--------------------------------|
| 1995-1999 | • Ph.D. in computer Science. |
| 1991-1994 | • E.N.S.T. engineering school. |
| 1988 | • Baccalaureus. |

Figure 1: CurVe sample output

how to use *styles* instead of *manually* formatting all paragraphs the same way). Hence, using a WYSIWYG editor *properly* turns out to be very close to using L^AT_EX itself. Since I already knew L^AT_EX, it would have been a waste of time to learn yet another tool.

For an interesting discussion on WYSIWYG solutions for L^AT_EX, see Kastrup (2002).

The next solution was to use an already existing class or style with L^AT_EX. For reasons that I won't mention here, but that can be found in its documentation's introduction, the only reasonable option was **currvita** (Reichert, 1999). This solution did not satisfy me, mainly because the layout was too limited for me. Layout is probably something very personal in this context, but I also believe that there are different traditions in different countries, and that CurVe provides a layout that pleases many people, notably in France. The increasing number of people using CurVe tends to show that I am not mistaken.

So I decided to use L^AT_EX from scratch (some other reasons for this are explained later). Being a lazy kind of person however, I tried *not* to reinvent the wheel, and I found that David Carlisle's L^AT_EXtable package would do most of the formatting for me. Being also a free software kind of person, I wanted other people to benefit from my work, so I naturally made it a proper class with complete documentation.

The first release of **CurVe** occurred in February 2001. Today, **CurVe** is a simple yet powerful class, consisting of only 350 lines of code, probably half of which is devoted to handling user customizations.

2 Layout

The primary purpose of **CurVe** is to offer a set of pre-defined commands to specify the contents of your CV, while removing from you the burden of formatting it. This has two important consequences, however: **CurVe** requires that you conform to its document structuring scheme, and will expect that you like the way it formats the output.

Figure 1 shows the output of **CurVe** on a very small sample CV. There is no user-level customization in this example.

2.1 Headers

As you can see in the example, a **CurVe** CV begins with two optional headers (upper left and upper right) in which you usually put your name, address, email, whether you're married and so on. These headers will respectively be left and right aligned. **CurVe** also lets you insert a small identity photo in the headers, either on the left, on the right, or between them. After these headers comes an optional title and an optional subtitle, which will be centered on the page.

The headers and titles in this example are specified in the document's preamble as follows:

```
\leftheader{%
 \textbf{Didier Verna} \\
 Born April 21$^{\text{st}}$ 1970 \\
 French}

\righthead{%
 \url{mailto:didier@lrde.epita.fr} \\
 \url{http://www.lrde.epita.fr/~didier} }

\title{Curriculum Vit\ae}
\subtitle{Sample made with CurVe}
```

Later on, after the call to `\begin{document}`, these headers are formatted like this:

```
\makeheaders[t]
\maketitle
```

This scheme is very traditional in L^AT_EX classes and should not surprise anybody. The optional argument to `\makeheaders` specifies the vertical alignment. Here, `top` is used. Many more commands are available to customize the appearance of the headers (positioning, size, fonts, etc.).

2.1.1 Rubrics

The remainder of the document is composed of sections called “rubrics” in the **CuVe** terminology. A rubric represents a major topic that you want to detail in your CV. Typical rubrics, as demonstrated in the example, are “Education”, “Professional Experience” and the like. Rubrics have a title (which will be centered) and appear under the form of properly aligned “entries” (see below). If a rubric has to be split across different pages, its title will be repeated automatically.

Here is the code used to generate the “Professional Education” rubric:

```
\begin{rubric}{Professional Experience}
\subrubric{Lecturing}
\entry*[2002-...] \textbf{\LaTeXe: an
    overview}. 3 hours conference.
\entry*[2000-...] \textbf{OpenGL Programming}.
    15 hours lecture, including workshops.
\entry* \textbf{Operating Systems}.
    30 hours lecture.

\subrubric{Development}
\entry*[\LaTeXe] Author of CuVe, FiNK and
    FiXme.
\entry*[XEmacs] Member of the review board.
\entry*[GNU] Contributor to other free
    software projects.
\end{rubric}
```

As you can see, each rubric is made within a **rubric** environment, which takes the rubric’s title as a mandatory argument. The other commands will be explained below. Please note that for technical reasons due to the use of the **LTXtable** package, each rubric must be written in a separate file. This is not a heavy burden however, and it even made things easier when I implemented the “flavor” mechanism described in the next section.

2.1.2 Subrubs

You might want to further split your rubrics into different “subrubs”. For instance, figure 1 shows two subrubs named “Lecturing” and “Development” under the “Professional Experience” rubric. Subrubs’ names are displayed in alignment with the entries’ contents (see below), but are formatted differently so that they remain distinguishable. Subrubs are created with the **\subrubric** command. For instance:

```
\subrubric{Development}
```

2.1.3 Entries

An entry is a final item of information related to the (sub)rubric under which it appears. An entry has a “contents”, and an optional “key” under which it is classified. For instance, consider the “XEmacs” entry in the example:

```
\entry*[XEmacs] Member of the review board.
```

The key is “XEmacs” and the entry’s contents is “Member of the review board.”. Keys are usually used to indicate dates. **CuVe** aligns both keys and contents together. Keys are optional (hence, they should be input within brackets) in order for you to classify several entries together (without repeating the same key over and over again). For instance:

```
\entry*[2000-...] \textbf{OpenGL Programming}.
    15 hours lecture, including workshops.
\entry* \textbf{Operating Systems}.
    30 hours lecture.
```

Here, the second key is not repeated because the “Operating Systems” course began in the same year as the OpenGL one.

You probably have noticed already that a “starified” version of the **\entry** command is used. Actually, **\entry*** commands in **rubric** environments are very similar to **\item** commands in **list** environments. The reason for this “starification” is that the first version of **CuVe** had a somewhat ill-designed, unstarred **\entry** command that took the whole entry’s contents as a second argument. The new scheme is much more elegant, but backward compatibility has been preserved.

3 The “flavor” mechanism

This is a very nice feature of **CuVe**, and one of the reasons that made me write it in the first place.

It is often desirable to maintain several slightly divergent versions of one’s CV at the same time. For instance, when I was looking for a job back in 2001, I had a version of my CV emphasizing Artificial Intelligence, and another emphasizing Distributed Virtual Reality. Only the title and some entries in the “Professional Experience” rubric were a bit different; all other parts basically remained the same. **CuVe** provides an easy-to-use mechanism for maintaining different “flavors” of your CV at the same time. You basically write different versions of (some of) your rubrics in different files (this is needed for technical reasons anyway, as mentioned previously), tell **CuVe** which flavor you want to format (**CuVe** can even ask you which one to use directly). **CuVe** will then use the global skeleton, and whenever it finds a rubric file specialized for that particular flavor, it

will use it. Otherwise, it will simply fall back to the default one (no particular flavor).

With a clever use of Makefiles, you can even manage to compile different flavors at the same time. Suppose you have a mainstream CV (let's call it `cv.tex`), with an "education" rubric in a file named `education.tex` and a "skills" one in `skills.tex`. Suppose further that you want a slightly divergent version of your CV with an alternate skills rubric.

The normal way to compile the alternate version is to save a new rubric file called, for instance, `skills.alt.tex`, and tell **CurVe** to use it by calling `\flavor{alt}` in your document's preamble. The only problem is that this alternate version will also compile to `cv.dvi` because both versions share the same skeleton and other rubric files.

To remedy this problem, you can make **CurVe** ask you at run-time which flavor to compile (this is done with the `ask` class option) and use special Makefile rules to answer the question for you, build the different flavors and move the different output files to flavor-specific names. Here is a Makefile example that would allow you to do this for as many different versions of `skills.tex` as you wish:

```
FLAVORS = alt # make other flavors if you want

all: all_flavors cv.dvi

all_flavors:
    for i in $(FLAVORS) ; do \
        $(MAKE) cv.$i.dvi FLAVOR=$$i ; \
    done

cv.$(FLAVOR).dvi: cv.tex education.tex \
    skills.$(FLAVOR).tex
    echo $(FLAVOR) | latex cv.tex
    mv cv.dvi $@

cv.dvi: cv.tex education.tex skills.tex
    echo | latex cv.dvi
```

As you can see, the trick is to fork a new `make` subprocess for each flavor you want to build, and obviously finish with the mainstream version. An `echo` shell command is used in conjunction with the `ask` class option to indicate the current flavor to build. For each `make` subprocess, this flavor is stored in the dynamic variable `FLAVOR`, and the corresponding `cv.$(FLAVOR).dvi` rule is used, with the proper dependencies.

An implementation sidenote. In order to implement the flavor mechanism, the `LATEX` macro `\input` has been redefined to look for flavored files first. This is actually very nice because you can use

it to make different flavors of text that do not belong in rubrics. For instance, suppose you want a special version of the subtitle of your CV for the flavor `alt`. Put your alternate subtitle in a file called `subtitle.alt.tex`; do something similar for the default subtitle. Now go to the skeleton of your CV, and write `\input{subtitle}` in the preamble. That's it. You'll have different subtitles in your different CV flavors.

4 Conclusion

In this paper, I preferred to concentrate on special aspects of **CurVe**'s history or peculiarities rather than on its user interface, because the latter is relatively straightforward and fully described in the package documentation. Although the layout of **CurVe** is strict, it remains deeply customizable. **CurVe** also has other features that are worth mentioning, like support for all standard class options, support for the `.ltx` file extension if, like me, you prefer it over `.tex` for `LATEX` files, support for standard `LATEX` bibliographic commands (although I don't recommend using them), and support for AUC-`TEX`. Additionally, **CurVe** has been translated into six different languages (English, French, Spanish, German, Italian and Danish).

For an example of what you can do with **CurVe**, get my own CV at <http://www.lrde.epita.fr/~didier/perso/cv.php>. You will also find a page describing some formatting tricks I've used to make it more eye-catching.

Thanks to different contributors of code or suggestions, **CurVe** has evolved since its first release, and I hope it will continue to evolve and reach more and more users in the future.

References

- Kastrup, David. "Revisiting WYSIWYG paradigms for authoring LATEX". 2002. <http://preview-latex.sourceforge.net/wysiwyg-draft.pdf>.
- Reichert, Axel. "currvita.sty". 1999. A curriculum vitae style for LATEX, available in macros/latex/contrib on CTAN.

◊ Didier Verna
EPITA, Research and Development
Laboratory,
14-16 rue Voltaire
94276 Le Kremlin-Bicêtre,
France
`didier@lrde.epita.fr`
<http://www.lrde.epita.fr/~didier>

Abstracts

Les Cahiers GUTenberg
Contents of Double Issue 39/40
(May 2001)

DANIEL FLIPO, Éditorial : le document au XXI^e siècle [Editorial: The document in the 21st century]; pp. 5–6

These are the conference proceedings (although not all papers are included here) from GUTenberg's 2001 annual meeting, held in Metz, France, 14–17 May, 2001. The editor begins by recalling Michel Goossens' closing words to his editorial introducing the proceedings from EuroTEX'98 in Saint Malo:

To conclude, I would like to stress how the enthusiasm of the participants . . . has transformed EuroTEX'98 into a real TEX fiesta, proving once more that the Lion and Friends are well-prepared and ready to enter the next millennium with confidence and limitless energy! [*Cahiers* 28/29 March 1998, p. vii]

Flipo then lists the many avenues of ongoing steady progress since that time: pdfTEX, Ω , the annual TEX Live CDs, even a successor to TEX itself.

The remainder of the editorial focuses on the conference and its contributions to that same steady progress. There were three main themes: new TEX developments, electronic documents in education, and back-and-forth conversion between LATEX and XML/MathML. The volume concludes with an article by Hans Hagen regarding the lessons to be learned from the NTS experience.¹

FRÉDÉRIC BOULANGER, LATEX au pays des tableurs [LATEX in the land of spreadsheets]; pp. 7–16

We show that some spreadsheet-like documents can be handled efficiently with LATEX. The main advantage of this approach is the ability to design separately the computations and the layout of the document.

We begin with a flight log for private pilots, the layout of which is fixed and normalised. Then, we present a document class for paysheets, where some commands are used to define the structure of a paysheet, and a command allows us to edit it according to an already defined structure.

[Author's abstract (edited)]

¹ Originally appeared in *Die TEXnische Komödie*, no. 1 (2001), pp. 36–53. Also appeared in *TUGboat*, 22 (1/2), March/June 2001, pp. 58–66.

DANIEL TAUPIN, Les polices TTF converties en METAFONT [Converting TTF fonts into METAFONT]; pp. 17–21

Using the *ttf2mf* program by Oleg Motygin, TTF (TrueType) fonts usually available under Windows 9x/NT were converted to METAFONT. Thus, they are now usable under any environment and any TEX/LATEX 2 _{ε} distribution, provided that the METAFONT generator is available, which is usually the case.

In addition, the availability of a symbolic source makes it possible to correct imperfections and to eventually create some missing characters.

[Author's abstract (edited)]

JEAN-MICHEL HUFFLEN, Vers une extension multilingue de BIBTEX [Towards a multilingual BIBTEX extension]; pp. 23–38

This paper describes a multilingual extension to BIBTEX, and how it was implemented. We aimed to meet such user requirements as having the language used for the bibliographical references of a printed work to be in the work's language, as well as the option of having the language of each reference to be that of the reference. Our tool will be able to work with any multilingual LATEX 2 _{ε} package. In addition, our extension allows both existing bibliographical files (.bib files) and bibliography style files (.bst files) to be processed compatibly.

[Author's abstract (edited)]

JORIS VAN DER HOEVEN, GNU TEXMACS: A free, structured WYSIWYG and technical text editor; pp. 39–50

There is a common belief that WYSIWYG technical editors are not suited for editing structured texts and generating documents of high typographic quality. In this paper, we analyze the reasons behind this belief. We next discuss the program GNU TEXMACS and some of its innovations in relation to the difficulties of structured, WYSIWYG, technical text editing.

[Author's abstract (edited)]

AZZEDDINE LAZREK, Aspects de la problématique de la confection d'une fonte pour les mathématiques arabes [Aspects to creating a font for mathematics in Arabic texts]; pp. 51–62

A good deal of Arabic mathematics texts contain formulas composed with specific symbols set in a text that runs from right to left. Up till now, as far as we know, there has been no system that makes the typesetting of such documents possible. Millions of scholars throughout Arabic countries use handbooks where symbols are still written in by hand.

The system presented in this paper is an attempt to provide the possibility of typesetting such

documents. The capabilities of both *TEX* and *ArabTEX* will be extended to suit this situation.

The Arabic font Naskh designed by K. Lagally for his package *ArabTEX* and the Computer Modern family designed by D. E. Knuth in *METAFONT* ... serve as the basic fonts. In math mode, Naskh is adapted to various sizes according to the different positions (normal, super- or subscript, etc.). This font is used to design new signs in different shapes and for abbreviations. The Computer Modern family, especially the Math Symbols and Math Extension fonts, will help build some special symbols via glyph inversion. Many difficulties arise afterwards: heterogeneity of size, bold face level, the position of symbols with respect to the baseline ... shape changes in Arabic characters when passing from text to math mode, to name a few. These difficulties attest to the limits of systems composed for the needs of typesetting mathematica in a Latin language whenever these systems are to be adapted to foreign language contexts.

[Author's abstract (edited)]

JEAN-MICHEL SARLAT and JEAN-PAUL VIGNAULT, *TEX dans l'enseignement secondaire, une expérience* [*TEX* in secondary education: An experiment]; pp. 63–69

We present some issues—established facts, thoughts, wishes—with respect to our efforts to introduce *TEX* to secondary school teachers, as well as providing a server to consolidate various initiatives.

[Translation of French résumé]

YOLAINÉ BOURDA, *Objets pédagogiques, vous avez dit objects pédagogiques ?* [Learning objects ... you said 'learning objects' ?]; pp. 71–79

Learning objects are currently the subject of numerous standardization projects. Unfortunately, the definition itself of 'learning objects' is still quite fluid.

This article attempts to look at what is meant by learning objects and to ask questions about their level of granularity and structuring.

[Translation of French résumé]

LAURENT ROMARY, *Un modèle abstrait pour la représentation de terminologies multilingues informatisées TMF—Terminological Mark-up Framework* [An abstract model for representing computerized TMF (Terminological Mark-up Framework) multilingual terminology]; pp. 81–88

We introduce an abstract model for representing computerized multilingual terminologies. This model was developed in XML by Technical Committee 37 of ISO. It relies on a methodology which makes an essential distinction between the general

structure of a terminological database and the information units (data categories) that are used to describe the various levels of this structure.

[Author's abstract (edited)]

ÉRIC-OLIVIER LOCHARD and DOMINIQUE TAURISSON, "Le monde selon Arcane": un paradigme instrumental pour l'édition électronique ["The world according to Arcane": An instrumental paradigm for electronic editing]; pp. 89–105

The World According to Arcane is an operating instrumental paradigm for electronic editing of scientifically established texts and knowledge, currently being used in several scholarly edition projects. The world of knowledge is edited in a database, the architecture of which is both generic (so as to be applicable to numerous domains) and simple (any information is a subject of interest): a multimedia document, a relation between subjects or an enrichment. Internal or external documents are enriched with the editing module, independently of the media and the final publication. The reading module offers very powerful procedures to investigate and browse electronic work: typified links inferred by the architecture, sophisticated indexation, dynamic composition of virtual documents, naturally formulated requests, formal treatments, and reading itineraries. The publishing module allows one to export information in various formats (HTML, XML, *TEX*), to compose paper books, and to produce electronic books in the form of autonomous applications distributed on CD-ROM, DVD-ROM, web site, or database system.

[Author's abstract (edited)]

DENIS ROEGEL, *La géométrie dans l'espace avec METAPOST* [Geometry in space with *METAPOST*]; pp. 107–138

METAPOST is a tool especially well suited for the inclusion of technical drawings in a document. In this article, we show how *METAPOST* can be used to represent objects in space and especially how it can be used for drawing geometric constructions involving lines, planes, as well as their intersections, orthogonal planes, etc. All the features belong to a new *METAPOST* package aimed at all those who teach and study geometry.

[Author's abstract]

YANNIS HARALAMBOUS and JOHN PLAICE, *Traitements automatiques des langues et composition sous Omega* [Natural language processing and composition under Omega]; pp. 139–166

While Ω continues to evolve and its functionality expand and diversify, one notices that the

methods used to make it possible to typeset Oriental languages can also be used to resolve problems left unanswered in Occidental languages. The same types of tools that break Thai phrases down into words and then syllables can also be used to determine whether a letter ‘s’ in a German word written in Gothic ought to be long or short. In these two cases, the tool in question is a *morphological analyzer*, often used in a field of study known as Natural Language Processing (*traitement automatique des langues* in French). Thanks to Ω ’s external OTP (Omega Translation Process), we can integrate such tools into Ω and use them in real time during composition.

In this article we will study six instances where such tools or linguistic methods are used, each varying in complexity and covering a broad range of languages: English, German, Greek, Arabic, Thai and Japanese.

[Translation of text drawn from opening paragraphs]

JEAN-PAUL JORDA, MARIE-LOUISE CHAIX and AHMED MAHBOUB, L^AT_EX et XML dans la chaîne éditoriale d’EDP Sciences [L^AT_EX and XML in the production process at EDP Sciences]; pp. 167–179

Recent developments in the use of L^AT_EX and related tools at EDP Sciences are presented. The production process of the journal *Astronomy and Astrophysics* is described, along with the production and use of XML files generated from L^AT_EX file headers.

[Author’s abstract]

ANDRÉ VIANTE, Une solution de conversion RTF vers XML/MathML avec publication Web dynamique en XML/MathML [RTF-to-XML/MathML conversion with dynamic Web publication also in XML/MathML]; pp. 181–200

There are several ways to generate HTML web pages containing mathematical material. For this purpose, one almost always needs to statically translate mathematical equations into images, applets or even plug-in data. However, updating the website then becomes an onerous task, while clients will often need special navigation environments.

A rather different approach is to publish “dynamically”, directly with XML/MathML—this is possible by using “server-side” technology.

In this presentation, we will present a solution that makes simple XML/MathML generation from RTF files possible, along with quick publication over the Internet (or other) without constraints on the client side. This can be achieved with Cocoon, XSLT and T_EX.

[Author’s abstract (edited)]

CHRIS ROWLEY, XSL FOs and T_EX: Some data; pp. 201–204

The XSL FO (Formatting Objects) specification is a noble and inventive project that is maturing into a cornucopia of useful insights and intellectual treats. How can it fail to delight when it formally describes basic properties, such as *visibility*, as ‘magic’!

It is therefore a timely, fascinating and pragmatic exercise to analyse the assumptions made by XSL about the process and results of document formatting. This article provides a small amount of the data needed for this analysis by comparing some aspects of the XSL model with that provided by T_EX/L^AT_EX.

[Author’s abstract]

HANS HAGEN, The status quo of the NTS project; pp. 205–220

A report on the NTS project was presented by Phil Taylor at the previous GUTenberg annual meeting in Toulouse [May 2000; report in *Cahier 35–36*, pp. 53–78]. Hans Hagen, who had been charged by Dante with making an independent audit report on the project, herein presents his views—fairly critical—on the manner in which the project had been conducted; he then proposes to regroup the efforts of the NTS, Ω and pdfT_EX developers to give T_EX the successor it deserves.

[Translation of Editor’s Note]

— * —

Articles from *Cahiers* issues can be found in PDF format at the following site:

<http://www.gutenberg.eu.org/pub/gut/publications>

[Compiled by Christina Thiele]

The most exciting \TeX event of 2002, the international conference of \TeX Users Group, is scheduled to be conducted in India during September 4-7, 2002 at Technopark, Trivandrum, the capital of the South Indian State of Kerala, fondly called *God's Own Country* for its wealth of natural charms.

Themes for Conference

- Using \TeX to typeset XML
- Multilingual typesetting using Omega
- High quality hyperdocuments using $\text{pdf}\text{\TeX}$
- Fonts for non-Latin languages
- New directions for Metafont and Metapost



23rd Annual Meeting and Conference

September 4-7, 2002, Technopark, Trivandrum, Kerala, India

Trumpet loudly TUG 2002's theme, all ye faithful:
Stand up and be proud of \TeX !



Important Dates

- Jan/Feb 2002: Send in abstracts for papers
- 28 Feb 2002: Notification of acceptance of paper
- 31 Mar 2002: Preliminary program available
- May 2002: Send first version of full paper
- July 2002: send final version of full paper
- 1-3 September 2002: Pre-conference tutorial in India
- 4-7 September 2002: TUG conference in India

Tutorial (1-3 September 2002)

- Introduction to \TeX
- \LaTeX to SGML/XML/MathML conversion
- XML and XSL transformation procedures
- TEI XML
- Generation of hyperlinked documents with $\text{pdf}\text{\TeX}$ and $\text{Con}\text{\TeXt}$
- Multilingual typesetting using Omega

Subscribe to TUG2002 Mailing List here

- URL: <http://www.tug.org.in/tug2002>
- General: tug2002@tug.org.in
- Travel: travel@tug2002.tug.org.in
- Abstracts: papers@tug2002.tug.org.in

Calendar

2002

- Feb 20–23 DANTE 2002, 26th meeting, Universität Erlangen-Nürnberg, Germany. For information, visit <http://www.dante.de/dante2002/>.
- Apr 29– May 3 EuroBachotEX 2002, 13th meeting of European TeX Users and 10th annual meeting of the Polish TeX Users' Group (GUST), "TeX and beyond", Bachotek, Brodnica Lake District, Poland. For information, visit <http://www.gust.org.pl/BachotEX/2002/>.
- May 29 Journée GUTenberg, "Distributions", Paris, France. For information, visit <http://www.gutenberg.eu.org/>.
- Jul 12–14 TypeCon 2002, Toronto, Canada. For information, visit <http://www.typecon2002.com>.
- Jul 21–26 SIGGRAPH 2002, San Antonio, Texas. For information, visit <http://www.siggraph.org/calendar/>.

TUG 2002

- International Convention Centre, Trivandrum, India. For information, visit <http://www.tug.org.in/tug2002/>.
- Sep 1–3 Tutorials: L^AT_EX; L^AT_EX to XML; METAPOST; the Text Encoding Initiative; TeX macro expansion.
- Sep 4–7 The 23rd annual meeting of the TeX Users Group, "Stand up and be proud of TeX!". For information, visit <http://www.tug.org.in/tug2002/>.

- Sep 9–13 Seybold San Francisco, San Francisco, California. For information, visit <http://www.key3media.com/seyboldseminars/events/events.shtml>.

- Sep 19–22 Association Typographique Internationale (ATypI) annual conference, Rome, Italy. For information, visit <http://www.atypi.org/rome2002/>.
- Sep 24–25 First Annual Conference, Friends of St. Bride Printing Library, London, England. For information, visit <http://www.stbride.org/conference.htm>
- Oct 4–5 DANTE 2002, 27th meeting, Zentrum Universität Augsburg, Germany. For information, visit
- Oct 12 UK TUG Autumn meeting, Nottingham University. For information, contact Dick Nickalls, dicknickalls@compuserve.com.
- Oct 14–17 Book History Workshop, Institut d'histoire du livre, Lyons, France. For information, visit <http://ihl.enssib.fr>.
- Oct 20–23 Conférence Fédérative sur le Document, Hammamet, Tunisia. For information, visit <http://www.loria.fr/conferences/cfd/>.
- Nov 8–9 ACM Symposium on Document Engineering, McLean, Virginia. For information, visit <http://www.documentengineering.org>.
- Nov 21 NTG 30th meeting, Technische Universiteit Delft, Netherlands. For information, visit <http://www.ntg.nl/bijeen/bijeen30.html>.

2003

- Mar 24–28 IUC23, The 23rd Internationalization and Unicode Conference, "Unicode, Internationalization, the Web: The Global Connection", Prague, Czech Republic. For information, visit <http://www.unicode.org/iuc/iuc23/>.

Status as of 1 July 2003

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.

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

Owing to the lateness of this issue, please consider that all events shown for 2002 are included only "for the record".

Apr 2–4	DANTE 2003, 28 th meeting, Universität Bremen, Germany. For information, visit http://www.dante.de/dante2003/ .	
May 1–3	BachoTeX 2003, 11 th annual meeting of the Polish TeX Users' Group (GUST), Bachotek, Brodnica Lake District, Poland. For information, visit http://www.gust.org.pl/BachoTeX/2003/ .	
May 15–17	Typo Berlin 2003, the 8 th International Design Conference, Berlin, Germany. For information, visit http://www.typo-berlin.de .	
May 22	NTG 31 st meeting, Hogeschool Helicon, Zeist, Netherlands. For information, visit http://www.ntg.nl/bijeen/bijeen31.html .	
May 28–30	Society for Scholarly Publishing, 25 th annual meeting, "Navigating Change", Baltimore, Maryland. For information, visit http://www.sspnet.org .	
May 29–Jun 2	ACH/ALLC 2003: Joint International Conference of the Association for Computers and the Humanities, and Association for Literary and Linguistic Computing, "Web X: A Decade of the World Wide Web", University of Georgia, Athens, Georgia. For information, visit http://www.english.uga.edu/webx/ or the organization web site at http://www.ach.org .	
Jun 11–13	Seybold Seminars PDF Summit, Amsterdam, Netherlands. For information, visit http://www.seyboldseminars.com/pdf_summit/ .	
Jun 24–27	EuroTeX 2003, "Back to Typography", Brest (Brittany), France. For information, visit http://omega.enstb.org/eurotex2003/ . (The EuroTeX 2003 proceedings will be published in <i>TUGboat</i> .)	
Jul 7–Aug 8	Rare Book School Summer Session, University of Virginia, Charlottesville, Virginia. A series of one-week courses on topics concerning rare books, manuscripts, the history of books and printing, and special collections. For information, visit http://www.virginia.edu/oldbooks .	
Jul 17–20	TypeCon2003, "Counter Culture", Minneapolis, Minnesota. For information, visit http://www.typecon2003.com .	
<hr/>		
TUG 2003		
Outrigger Waikoloa Beach Resort, Big Island, Hawai'i.		
Jul 20–24	The 24 th annual meeting of the TeX Users Group, "Silver Anniversary—25 years!—of TeX". For information, visit http://www.tug.org/tug2003/ .	
Jul 27–Aug 1	SIGGRAPH 2003, San Diego, California. For information, visit http://www.siggraph.org/calendar/ .	
Aug 3	Web Document Analysis workshop, Edinburgh, Scotland, UK. For information, visit http://www.csc.liv.ac.uk/~wda2003 .	
Aug 3–6	ICDAR 2003, International Conference on Document Analysis and Recognition, Edinburgh, Scotland, UK. For information, visit http://www.essex.ac.uk/ese/icdar2003/ .	
Sep 3–5	24 th Internationalization and Unicode Conference (IUC24): "Unicode, Internationalization, the Web: Powering Global Business", Atlanta, Georgia. For information, visit http://www.unicode.org/iuc/iuc24/ .	
Sep 8–9	DANTE 29 th meeting, Universität Giessen, Germany. For information, visit http://www.dante.de/events/ .	
Sep 22–25	Seybold San Francisco, San Francisco, California. For information, visit http://www.seyboldseminars.com/sf2003/ .	
Sep 19–22	Association Typographique Internationale (ATypI) annual conference, "Between Text and Reader", Vancouver, Canada. For information, visit http://www.atypi.org/40_conferences .	
Oct 20–21	Second Annual St. Bride Conference, "Hidden Typography", London, England. For information, visit http://www.stbride.org/conference2003 .	
Nov 13	NTG 32 nd meeting, Arnhem, Netherlands; no details yet.	
Nov 20–22	ACM Symposium on Document Engineering, Grenoble, France. For information, visit http://www.documentengineering.org .	

TUG 2003: the Silver Anniversary – 25 years! – of TeX

The 24th Annual Meeting and Conference of the TeX Users Group

tug2003@tug.org

Themes

- Resurgence of TeX & L^AT_EX
- pdfTeX, ConTeXt, Metapost, Metafont
- TeX-XML Symbiosis, TEI, Digital Archiving
- Fonts & Graphics
- Installations & Management, CTAN
- Publisher & Prepress Dilemmas
- MacOS X TeX: New Kid on the Block

Important Dates

2002		2003	
Abstracts due	18 Nov	First draft of paper due	9 Feb
Abstracts accepted	18 Dec	Registration deadline	9 Apr
Early-lion registration	31 Dec	Final paper due	9 Jun

Links

- Homepage <http://www.tug.org/tug2003/>
News Mailing List <http://www.tug.org/tug2003/news/>
TeX Heritage <http://www.tug.org/tug2003/heritage/>
Call for Abstracts <http://www.tug.org/tug2003/callfor.html>
Registration/Donations <https://www.tug.org/tug2003/registration.html>



TeX Enthusiasts worldwide are invited to join us for a grand reunion to celebrate the accomplishments of TeX!
Polish up the old and learn about the shiny new directions in TeX!

July 20-24, 2003, Outrigger Waikoloa Beach Resort, Big Island, Hawaii
[Kona International Airport at Keahole (KOA)]

Institutional Members

American Mathematical Society,
Providence, Rhode Island

Center for Computing Science,
Bowie, Maryland

Cessna Aircraft Company,
Wichita, Kansas

The Clarinda Company,
Clarinda, Iowa

CNRS - IDRIS,
Orsay, France

CSTUG, *Praha, Czech Republic*

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

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

Institute for Advanced Study,
Princeton, New Jersey

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

Iowa State University,
Computation Center,
Ames, Iowa

Kluwer Academic Publishers,
Dordrecht, The Netherlands

KTH Royal Institute of
Technology, *Stockholm, Sweden*

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

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

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

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

Università degli Studi di Trieste,
Trieste, Italy

Vanderbilt University,
Nashville, Tennessee

TEX USERS GROUP

Promoting the use of **TEX** throughout the world

mailing address:

P.O. Box 2311
Portland, OR 97208-2311 USA

shipping address:

1466 NW Naito Parkway,
Suite 3141
Portland, OR 97209-2820 USA

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

President: Mimi Jett
Vice-President: Arthur Ogawa
Treasurer: Donald W. DeLand
Secretary: Susan DeMerit

Information for TUG membership list

TUG uses the information you provide to mail you products, publications, notices, and (for voting members) official ballots, or in a printed or electronic membership list, available to TUG members only.

Note: TUG neither sells its membership list nor provides it to anyone outside of its own membership.

Allowing TUG to send you notices electronically will generally ensure that you receive them much earlier than the notice in printed form. However, if you would rather not receive TUG notices via electronic mail, please check the appropriate box.

Do not send me TUG notices via email .

TUG plans to prepare a printed or electronic membership list, available to TUG members only. If you would like a listing in such a publication, please check the appropriate box.

Please do include my information in a published members-only TUG directory .

Name: _____

Department: _____

Institution: _____

Address: _____

Phone: _____ Fax: _____

Email address: _____

Position: _____ Affiliation: _____

2003 TUG Membership Form

Rates for TUG membership and TUGboat subscription are listed below. Please check the appropriate boxes and mail payment (in US dollars, drawn on an United States bank) along with a copy of this form. If paying by credit card, you may fax the completed form to the number at left.

- 2003 TUGboat (Volume 24).
- 2003 CD-ROMs include **TEX Live 8** (2 disks) and Dante's CTAN 2003 (4 disk set).
- *Multi-year orders:* You may use this year's rate to pay for more than one year of membership.
- Orders received after 31 May, 2003: please add \$10 to cover the additional expense of shipping back issues of TUGboat and CD-ROMs.

	Rate	Amount
Annual membership for 2003 (TUGboat, CD-ROMs)	<input type="checkbox"/>	\$65
Student/Senior membership for 2003 (TUGboat, CD-ROMs)*	<input type="checkbox"/>	\$35
Shipping charge (add to the above if after 31 May, 2003)	<input type="checkbox"/>	\$10
Subscription for 2003 (TUGboat, CD-ROMs) (non-voting)	<input type="checkbox"/>	\$85
Institutional Membership for 2003 (TUGboat, CD-ROMs) (includes up to seven members)	<input type="checkbox"/>	\$500
Materials for 2002**		
TUGboat Volume 23	<input type="checkbox"/>	\$45
T<small>EX</small> Live 7 CD-ROM	<input type="checkbox"/>	\$5
2002 CTAN CD-ROMs	<input type="checkbox"/>	\$10
Voluntary donations		
General TUG contribution	<input type="checkbox"/>	_____
Contribution to Bursary Fund†	<input type="checkbox"/>	_____
Contribution to T<small>EX</small> Development Fund ‡	<input type="checkbox"/>	_____
Total \$		_____

Payment (check one) Payment enclosed Charge Visa/Mastercard/AmEx

Account Number: _____

Exp. date: _____ Signature: _____

* Please attach photocopy of (if student) 2003 student ID or (if senior) ID showing age 65 years or older.

† The Bursary Fund provides financial assistance for attendance at the TUG Annual Meeting.

‡ The **TEX Development Fund** provides financial assistance for technical projects.

** If you were not a TUG member in 2002 and wish to receive **TEX Live** and CTAN CDs right away, these items are available with the purchase of a current 2003 membership.

TeX Consulting & Production Services

Information about these services can be obtained from:

TeX Users Group
 1466 NW Naito Parkway, Suite 3141
 Portland, OR 97209-2820, U.S.A.
 Phone: +1 503 223-9994

Fax: +1 503 223-3960
 Email: office@tug.org
 URL: <http://www.tug.org/consultants.html>

North America

Loew, Elizabeth

President, TeXniques, Inc.,
 675 Massachusetts Avenue, 6th Floor,
 Cambridge, MA 02139;
 (617) 876-2333; Fax: (781) 344-8158
 Email: loew@texniques.com

Complete book and journal production in the areas of mathematics, physics, engineering, and biology. Services include copyediting, layout, art sizing, preparation of electronic figures; we keyboard from raw manuscript or tweak TeX files.

Ogawa, Arthur

40453 Cherokee Oaks Drive,
 Three Rivers, CA 93271-9743;
 (209) 561-4585
 Email: Ogawa@teleport.com

Bookbuilding services, including design, copyedit, art, and composition; color is my specialty. Custom TeX macros and L^AT_EX2_E document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in L^AT_EX, 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 L^AT_EX. I have authored several popular packages for L^AT_EX 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 Unicorn Collaborative, Inc, Ted Zajdel

115 Aspen Drive, Suite K
 Pacheco, CA 94553
 (925) 689-7442
 Email: contact@unicorn-collab.com

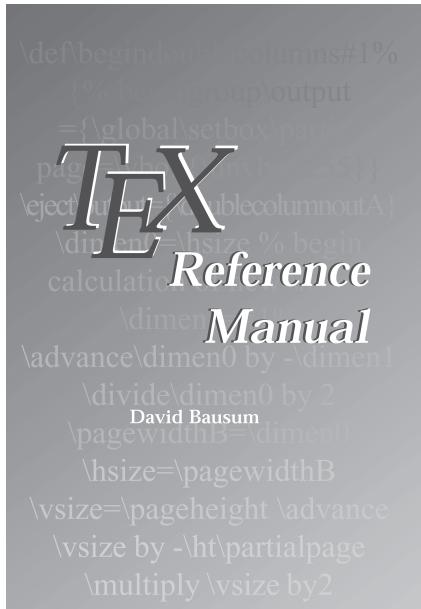
We are a technical documentation company, initiated in 1990, which time, strives for error free, seamless documentation, delivered on time, and within budget. We provide high quality documentation services such as document design, graphic design and copy editing. We have extensive experience using tools such as FrameMaker, TeX, L^AT_EX, Word, Acrobat, and many graphics programs. One of our specialties is producing technical manuals and books using L^AT_EX and TeX. Our experienced staff can be trained to use any tool required to meet your needs. We can help you develop, rewrite, or simply copy-edit your documentation. Our broad experience with different industries allows us to handle many types of documentation including, but not limited to, software and hardware systems, communications, scientific instrumentation, engineering, physics, astronomy, chemistry, pharmaceuticals, biotechnology, semiconductor technology, manufacturing and control systems. For more information see our web page <http://www.unicorn-collab.com>.

Outside North America

DocuTeXing: TeX Typesetting Facility

43 Ibn Kotaiba Street,
 Nasr City, Cairo 11471, Egypt
 +20 2 4034178; Fax: +20 2 4034178
 Email: main-office@DocuTeXing.com

DocuTeXing provides high-quality TeX and L^AT_EX typesetting services to authors, editors, and publishers. Our services extend from simple typesetting and technical illustrations to full production of electronic journals. For more information, samples, and references, please visit our web site: <http://www.DocuTeXing.com> or contact us by e-mail.



TeX Reference Manual

by **David Bausum**, Lighthouse & Associates, Beloit, WI, USA

The *TeX Reference Manual* is the first comprehensive reference manual written by a programmer for programmers. It contains reference pages for each of TeX's 325 primitive control sequences. Over 80% of its reference pages contain examples that range from simple to challenging. Each example is typeset verbatim in a style which is easy to read and experiment with. *TeX Reference Manual* also just typesets the example, so you can see what it makes, and explains how the example works. The description on each primitive's reference page is an annotated discussion of *The TeXbook*'s treatment of the primitive. That means a TeX user will find it natural to move back and forth between the two books. One of *TeX Reference Manual*'s innovative features is families. They simplify the search for the primitive which performs a particular task.

Primitive Control Sequences			
Family Name	Type	Description	
Box (29)	Logic (20)	C	Command (163)
Character (16)	Macro (20)	D	Derived Command (17)
Debugging (25)	Marks (4)	IQ	Internal Quantity (42)
File I/O (13)	Math (69)	PI	Parameter (integer) (55)
Fonts (5)	Page (13)	PD	Parameter (dimen) (21)
Glue (12)	Paragraph (30)	PG	Parameter (glue) (15)
Hyphenation (11)	Penalties (12)	PM	Parameter (muglue) (3)
Inserts (8)	Registers (11)	PT	Parameter (token) (9)
Job (11)	Tables (9)		
Kern (7)			

CONTENTS

Preface

1. Families and Primitive Control Sequences.
 2. Reference Pages for the Primitives.
- Appendix A.** Typesetting Verbatim Material.
Appendix B. Working with PostScript Fonts.
Appendix C. Typesetting Material in Two Columns.
 Bibliography. Index.

February 2002 Hardbound, ISBN 0-7923-7673-0 390 pp.
 EUR 108.00 / USD 99.00 / GBP 68.00
Special Price offered to TUGBOAT subscribers:
 EUR 97.00 / USD 90.00 / GBP 61.00

TeX Reference Manual has appendices which provide a comprehensive discussion of: verbatim material, PostScript fonts, and two-column material. In particular, one word describes its font macros, elegant. The *TeX Reference Manual* is an invaluable tool for both the experienced and new users of TeX.



ORDER TODAY!

ONLINE: WWW.WKAP.NL

Fax your order:
 USA: 781-681-9045
 Rest of World: +31 78 6546 474

Phone:
 USA: +781-871-6600
 Rest of World: +31 78 6392 392

Email:
 USA: kluwer@wkap.com
 Rest of World: services@wkap.nl

TEX **USERS** **GROUP**

Promoting the use of TEX throughout the world

mailing address:

P.O. Box 2311
Portland, OR 97208-2311 USA

shipping address:

1466 NW Naito Parkway,
Suite 3141
Portland, OR 97209-2820 USA

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

President: Mimi Jett
Vice-President: Arthur Ogawa
Treasurer: Donald W. DeLand
Secretary: Susan DeMerit

Information for TUG membership list

TUG uses the information you provide to mail you products, publications, notices, and (for voting members) official ballots, or in a printed or electronic membership list, available to TUG members only.

Note: TUG neither sells its membership list nor provides it to anyone outside of its own membership.

Allowing TUG to send you notices electronically will generally ensure that you receive them much earlier than the notice in printed form. However, if you would rather not receive TUG notices via electronic mail, please check the appropriate box.

Do not send me TUG notices via email .

TUG plans to prepare a printed or electronic membership list, available to TUG members only. If you would like a listing in such a publication, please check the appropriate box.

Please do include my information in a published members-only TUG directory .

Name: _____

Department: _____

Institution: _____

Address: _____

Phone: _____ Fax: _____

Email address: _____

Position: _____ Affiliation: _____

2003 TUG Membership Form

Rates for TUG membership and TUGboat subscription are listed below. Please check the appropriate boxes and mail payment (in US dollars, drawn on an United States bank) along with a copy of this form. If paying by credit card, you may fax the completed form to the number at left.

- 2003 TUGboat (Volume 24).
- 2003 CD-ROMs include TEX Live 8 (2 disks) and Dante's CTAN 2003 (4 disk set).
- *Multi-year orders:* You may use this year's rate to pay for more than one year of membership.
- Orders received after 31 May, 2003: please add \$10 to cover the additional expense of shipping back issues of TUGboat and CD-ROMs.

	Rate	Amount
Annual membership for 2003 (TUGboat, CD-ROMs)	<input type="checkbox"/>	\$65
Student/Senior membership for 2003 (TUGboat, CD-ROMs)*	<input type="checkbox"/>	\$35
Shipping charge (add to the above if after 31 May, 2003)	<input type="checkbox"/>	\$10
Subscription for 2003 (TUGboat, CD-ROMs) (non-voting)	<input type="checkbox"/>	\$85
Institutional Membership for 2003 (TUGboat, CD-ROMs) (includes up to seven members)	<input type="checkbox"/>	\$500
Materials for 2002**		
TUGboat Volume 23	<input type="checkbox"/>	\$45
T <small>EX</small> Live 7 CD-ROM	<input type="checkbox"/>	\$5
2002 CTAN CD-ROMs	<input type="checkbox"/>	\$10
Voluntary donations		
General TUG contribution	<input type="checkbox"/>	_____
Contribution to Bursary Fund†	<input type="checkbox"/>	_____
Contribution to T <small>EX</small> Development Fund‡	<input type="checkbox"/>	_____
Total \$		_____

Payment (check one) Payment enclosed Charge Visa/Mastercard/AmEx

Account Number: _____

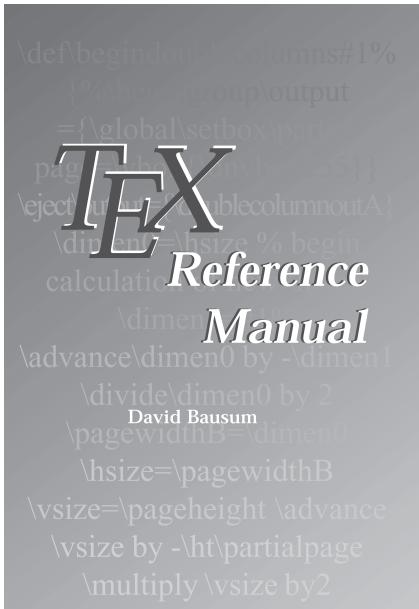
Exp. date: _____ Signature: _____

* Please attach photocopy of (if student) 2003 student ID or (if senior) ID showing age 65 years or older.

† The Bursary Fund provides financial assistance for attendance at the TUG Annual Meeting.

‡ The TEX Development Fund provides financial assistance for technical projects.

** If you were not a TUG member in 2002 and wish to receive TEX Live and CTAN CDs right away, these items are available with the purchase of a current 2003 membership.



TeX Reference Manual

by **David Bausum**, Lighthouse & Associates, Beloit, WI, USA

The *TeX Reference Manual* is the first comprehensive reference manual written by a programmer for programmers. It contains reference pages for each of TeX's 325 primitive control sequences. Over 80% of its reference pages contain examples that range from simple to challenging. Each example is typeset verbatim in a style which is easy to read and experiment with. *TeX Reference Manual* also just typesets the example, so you can see what it makes, and explains how the example works. The description on each primitive's reference page is an annotated discussion of *The TeXbook*'s treatment of the primitive. That means a TeX user will find it natural to move back and forth between the two books. One of *TeX Reference Manual*'s innovative features is families. They simplify the search for the primitive which performs a particular task.

Primitive Control Sequences			
Family Name	Type	Description	
Box (29)	Logic (20)	C	Command (163)
Character (16)	Macro (20)	D	Derived Command (17)
Debugging (25)	Marks (4)	IQ	Internal Quantity (42)
File I/O (13)	Math (69)	PI	Parameter (integer) (55)
Fonts (5)	Page (13)	PD	Parameter (dimen) (21)
Glue (12)	Paragraph (30)	PG	Parameter (glue) (15)
Hyphenation (11)	Penalties (12)	PM	Parameter (muglue) (3)
Inserts (8)	Registers (11)	PT	Parameter (token) (9)
Job (11)	Tables (9)		
Kern (7)			

CONTENTS

Preface

1. Families and Primitive Control Sequences.
 2. Reference Pages for the Primitives.
- Appendix A.** Typesetting Verbatim Material.
Appendix B. Working with PostScript Fonts.
Appendix C. Typesetting Material in Two Columns.
 Bibliography. Index.

February 2002 Hardbound, ISBN 0-7923-7673-0 390 pp.
 EUR 108.00 / USD 99.00 / GBP 68.00
Special Price offered to TUGBOAT subscribers:
 EUR 97.00 / USD 90.00 / GBP 61.00

TeX Reference Manual has appendices which provide a comprehensive discussion of: verbatim material, PostScript fonts, and two-column material. In particular, one word describes its font macros, elegant. The *TeX Reference Manual* is an invaluable tool for both the experienced and new users of TeX.



ORDER TODAY!

ONLINE: WWW.WKAP.NL

Fax your order:
 USA: 781-681-9045
 Rest of World: +31 78 6546 474

Phone:
 USA: +781-871-6600
 Rest of World: +31 78 6392 392

Email:
 USA: kluwer@wkap.com
 Rest of World: services@wkap.nl

introducing

TEXTURES® 2.0

WITH SYNCHRONICITY



AGAIN THE MACINTOSH DELIVERS A NEW TEX WITH A REVOLUTION IN HUMAN INTERFACE.

As computer power has advanced, the Macintosh has consistently been the leader in the human and humane connection to technology, and Textures has consistently led in bringing ease of use to TEX users.

First with Textures 1.0, the first truly

integrated TEX system. Then with Lightning Textures, the first truly interactive TEX system. Now, with Textures 2.0 and Synchronicity, Blue Sky Research again delivers a striking advance in TEX interactivity and productivity.

With Synchronicity, your TEX input documents are reliably and automatically cross-linked, correlated, or "synchronized" with the finished TEX typeset pages. Every piece of the finished product is tied directly to the source code from which it was generated, and vice-versa. To go from TEX input directly and exactly to the corresponding typeset characters, just click.

It's that simple: just click, and Textures will take you instantly and precisely to the corresponding location. And it goes both ways: just click on any typeset character, and Textures will take you directly to the TEX code that produced it. No matter how many input files you have, no matter what macros you use, Synchronicity will take you there, instantly and dependably.

I m p r o v e Y O U R p e r f o r m a n c e :

G E T S Y N C H R O N I C I T Y

BLUE SKY RESEARCH
317 SW ALDER STREET
PORTLAND, OR 97204 USA



800 622 8398
503 222 9571
WWW.BLUESKY.COM