MAPS

NUMMER 42 • VOORJAAR 2011

NEDERLANDSTALIGE TEX GEBRUIKERSGROEP

Voorzitter
Taco Hoekwater
ntg-president@ntg.nl

Secretaris
Willi Egger
ntg-secretary@ntg.nl

Penningmeester
Ferdy Hanssen
ntg-treasurer@ntg.nl

Bestuursleden
Frans Absil
fgj.absil@nlda.nl

Frans Goddijn
frans@goddijn.com

Hans Hagen
pragma@wxs.nl

Postadres
Nederlandstalige TEX Gebruikersgroep
Maasstraat 2
5836 BB Sambeek

ING bankrekening
1306238
t.n.v. NTG, Arnhem
BIC-code: INGBNL2A
IBAN-code: NL53INGB0001306238

E-mail bestuur
ntg@ntg.nl

E-mail MAPS redactie
maps@ntg.nl

WWW
www.ntg.nl

De **Nederlandstalige TEX Gebruikersgroep (NTG)** is een vereniging die tot doel heeft de kennis en het gebruik van TEX te bevorderen. De NTG fungeert als een forum voor nieuwe ontwikkelingen met betrekking tot computergebaseerde document-opmaak in het algemeen en de ontwikkeling van 'TEX and friends' in het bijzonder. De doelstellingen probeert de NTG te realiseren door onder meer het uitwisselen van informatie, het organiseren van conferenties en symposia met betrekking tot TEX en daarmee verwante programmatuur.
De NTG biedt haar leden ondermeer:

- Tweemaal per jaar een NTG-bijeenkomst.
- Het NTG-tijdschrift MAPS.
- De 'TEX Live'-distributie op DVD/CDROM inclusief de complete CTAN software-archieven.
- Verschillende discussielijsten (mailing lists) over TEX-gerelateerde onderwerpen, zowel voor beginners als gevorderden, algemeen en specialistisch.
- De FTP server `ftp.ntg.nl` waarop vele honderden megabytes aan algemeen te gebruiken 'TEX-producten' staan.
- De WWW server `www.ntg.nl` waarop algemene informatie staat over de NTG, bijeenkomsten, publicaties en links naar andere TEX sites.
- Korting op (buitenlandse) TEX-conferenties en -cursussen en op het lidmaatschap van andere TEX-gebruikersgroepen.

Lid worden kan door overmaking van de verschuldigde contributie naar de NTG-giro (zie links); vermeld IBAN- zowel als SWIFT/BIC-code en selecteer shared cost. Daarnaast dient via `www.ntg.nl` een informatieformulier te worden ingevuld. Zonodig kan ook een papieren formulier bij het secretariaat worden opgevraagd.
De contributie bedraagt € 40; voor studenten geldt een tarief van € 20. Dit geeft alle lidmaatschapsvoordelen maar *geen stemrecht.* Een bewijs van inschrijving is vereist. Een gecombineerd NTG/TUG-lidmaatschap levert een korting van 10% op beide contributies op. De prijs in euro's wordt bepaald door de dollarkoers aan het begin van het jaar. De ongekorte TUG-contributie is momenteel $85.

Afmelding kan met ingang van het volgende kalenderjaar door opzegging per e-mail aan de penningmeester.

MAPS bijdragen kunt u opsturen naar `maps@ntg.nl`, bij voorkeur in LATEX- of ConTEXt formaat. Bijdragen op alle niveaus van expertise zijn welkom.

Productie. De Maps wordt gezet met behulp van een LATEX class file en een ConTEXt module. Het pdf bestand voor de drukker wordt aangemaakt met behulp van pdftex 1.40.11 en luatex 0.70.1 draaiend onder Linux 2.6. De gebruikte fonts zijn Linux Libertine, het niet-proportionele font Inconsolata, schreefloze fonts uit de Latin Modern collectie, en de Euler wiskunde fonts, alle vrij beschikbaar.

TEX is een door professor Donald E. Knuth ontwikkelde 'opmaaktaal' voor het letterzetten van documenten, een documentopmaaksysteem. Met TEX is het mogelijk om kwalitatief hoogstaand drukwerk te vervaardigen. Het is eveneens zeer geschikt voor formules in mathematische teksten.
Er is een aantal op TEX gebaseerde producten, waarmee ook de logische structuur van een document beschreven kan worden, met behoud van de letterzet-mogelijkheden van TEX. Voorbeelden zijn LATEX van Leslie Lamport, $\mathcal{AMS}$-TEX van Michael Spivak, en ConTEXt van Hans Hagen.

Inhoudsopgave

Redactioneel

Vanochtend terwijl ik wakker werd regende het voor het eerst in twee maanden. En dat werd hoog tijd ook, want mijn tuinplanten begonnen er uit te zien als de achterkant van een droogboekettenwinkel met gebrek aan opslagruimte. Toch is het voorjaar voor mij altijd de beste periode van het jaar: als het weer net weer warm genoeg is om zonder jas even naar de winkel te rennen, je overal bloesems ruikt en ziet als je met een witbiertje 's middags nog even in de tuin zit, een moedereend met een grote verzameling kroost trots langs de parkvijver paradeert, de honden hun neus zo diep in de struikjes hebben dat een korte uitlaatsprint net zo lang duurt als een marathon in de winter, en de eerste Duitse toeristen met helmpjes op de stoep fietsend je bijna omver rijden voor je eigen voordeur, dan voel ik de nieuwe energie op me overspringen.

Dat is maar goed ook, want al die extra energie is hard nodig. Het voorjaar is een periode van deadline op deadline. Alles wat in de winter is blijven liggen moet dan toch nog even af, en daarbij komen dan nog de traditionele eens-per-jaar activiteiten: voorbereidingen voor BachoTEX en de voorjaarsbijeenkomst van de NTG, LuaTEX en MetaPost klaarmaken voor de TEXLive release, en natuurlijk ook de voorjaarsmaps. De Maps komt zeker niet in de laatste plaats, maar wel in de laatste *tijd* (wat de reden is dat wanneer je dit leest, de NTG bijeenkomst alweer voorbij is).

Dit is de eerste Maps die geproduceerd is bij de nieuwe printer, en een onverwacht voordeel daarvan is dat een proefexemplaar al naast me ligt terwijl ik dit redactioneel in zit te typen. Dat is mooi omdat ik nu al meteen zeker weet dat de figuren er goed in staan. In het oude proces had ik die luxe niet: de vorige drukker leverde geen proef maar een zogenaamde ‚uithouder'. Die werd dan weliswaar opgestuurd als losse vellen voordat het inlijmen en snijden werd gedaan, maar toch was de hele Maps dan al wel gedrukt. En vanwege de tijdsdruk betekende dat dat een foutje soms doelbewust niet gecorrigeerd werd. Een niet onbelangrijk voordeel, want aan de hand van de proef heb ik eerder vanochtend nog op anderhalf dozijn pagina's correcties doorgevoerd.

De achterzijde van deze pagina heeft traditiegetrouw een aankondiging. Deze keer voor de tug2011-bijeenkomst, die vanwege de onrust afgelopen tijd *niet* in Egypte plaatsvindt, maar in India. Ontbrekend is een aankondiging voor de vijfde ConTEXt-bijeenkomst, die dit jaar plaats vindt in Porquerolles, Frankrijk. De early-bird registratie sloot officieel op 31 mei, maar voor de lezers van deze Maps kunnen we nog wel een kleine uitzondering maken, tot een week nadat hij in de brievenbus is gevallen kun je je nog opgeven voor het verlaagde tarief. Zet er dan wel even een opmerking bij zodat we je niet later gaan lastigvallen. Zie http://meeting.contextgarden.net/ voor meer informatie en het opgaveformulier.

Terug naar de inhoud van de Maps. Twee besprekingen van Engelse vertalingen van boeken van Herbert Voß deze keer: één behoorlijk lovend, één behoorlijk negatief. Ik ben nieuwsgierig naar wat er zou gebeuren als de twee reviewers hun boeken zouden omruilen, dus misschien komt er in de volgende Maps nog wel een vervolg.

In deze tijd heeft misschien iedereen wel een smartphone voor het bijhouden van afspraken, maar het kan ook tegelijkertijd traditioneler en toch hipper: met een ‚PocketMod'. Willi legt in een artikel met voorbeelden uit hoe zijn ConTEXt module daarvoor werkt.

Tagged pdf is zo'n wonderwoord dat de laatste paar jaar opgang maakt. Waar het goed voor is weet vrijwel niemand, zelfs Hans niet, maar in ieder geval kun je het maken met ConTEXt. Ook van de hand van Hans is een kort stukje over letterspatiëring en ligaturen.

Twee artikelen van Kees vullen samen ongeveer de helft van deze Maps. Natuurlijk schrijft hij over PostScript, niet over TEX, maar dat maakt de artikelen niet minder interessant. Wel langzaam lezen en je niet laten afleiden door de mooie plaatjes, want er komt behoorlijk wat (wiskundige) informatie op je af.

Graham Douglas is helemaal weg van LuaTEX, en presenteert een setje Lua macros om de LaTEX paginalayout in te stellen aan de hand van een DTP-specificatie. Daarnaast nog artikelen van mezelf, Luigi Scarso, en Thomas Schmitz die ook allemaal over LuaTEX gaan, en met name over hoe je met behulp van externe Luamodules ConTEXt-dingen kunt laten doen die met een traditionelere TEX-engine volstrekt onmogelijk zouden zijn.

En daarmee zijn zowel de Maps als deze pagina vol. Rest mij nog te zeggen dat de deadline voor de volgende Maps 1 oktober 2011 is. Tot die tijd:

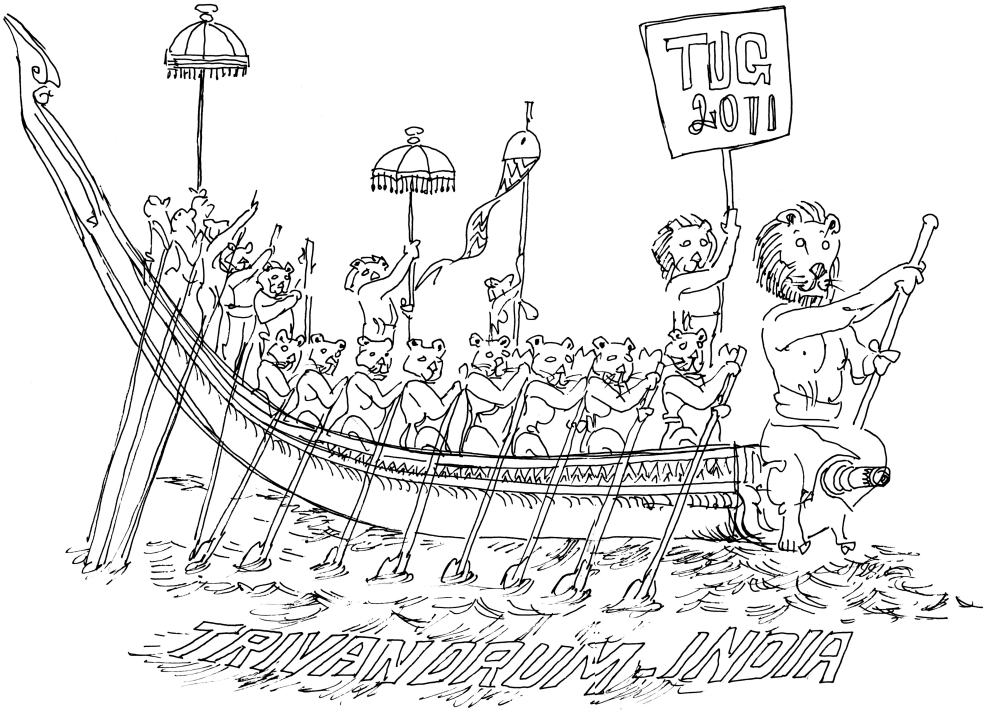
Veel leesplezier toegewenst,

Taco Hoekwater

TUG 2011: TEX in the eBook era

Presentations covering the TEX world
The 32nd Annual Meeting of the TEX Users Group

http://tug.org/tug2011 ■ tug2011@tug.org



October 19–21, 2011

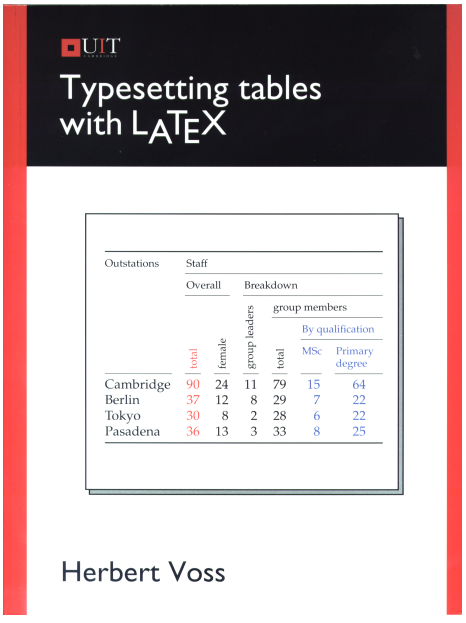
River Valley Technologies
Trivandrum, Kerala
India

- July 15, 2011 — presentation proposal deadline
- August 1, 2011 — early bird registration deadline
- September 1, 2011 — preprints deadline
- October 19–21, 2011 — conference and workshop
- October 31, 2011 — deadline for final papers

Sponsored by the TEX Users Group, DANTE e.V., and River Valley Technologies.

Review of *Typesetting tables with LaTeX* by Herbert Voss

LaTeX en tabellen: het is geen gemakkelijk duo. Van de beginneling vraagt het wat opzoekwerk en experimenteren, maar vooral heel wat concentratie bij het aanbrengen van de code.
Misschien heb je ooit overwogen om bij de aanmaak van tabellen je schaamteloos over te geven aan de eenvoud van een wysiwyg officepakket met LaTeX -exportfilter[1]. Wysiwyg lost echter niet alle problemen op, esthetisch kan het vaak beter maar het zorgt er wel voor dat de nodige ampersanden op de juiste plaats komen te staan.
The LaTeX Companion[2] besteedt welgeteld 41 bladzijden aan tabellen[3]. Dat is voldoende om op weg gezet te worden maar gemakkelijk wordt het niet en sommige topics komen gewoon niet aan bod in The LaTeX Companion: bijv. data-import vanuit een extern bestand.



'Typesetting tables with LaTeX ' door Herbert Voss is als volgt opgebouwd: een eerste introductie met de 'oude bekenden': tabular en tabbing. Snel gevolgd door een lijvig tweede hoofdstuk met de bespreking van maar liefst 29 pakketten. Kleur krijgt de aandacht in hoofdstuk 3. Tabellen over meerdere pagina's in hoofdstuk 4. Hoofdstuk 5 brengt tips en tricks. Om af te sluiten met een 36 bladzijden tellend hoofdstuk aan voorbeelden.

Hoofdstuk 1 moet je zeker gelezen hebben vooraleer je verder gaat met de andere. Alle basisinformatie met betrekking tot tabellen staat hierin vermeld. Er worden onder andere enkele tips meegegeven over de opbouw van tabellen en wat de oorzaak is van veel voorkomende fouten. Zowel voor beginner als gevorderde is dit zeker niet te missen.

Hoofdstuk 2 is geen pure opsomming van de 29 pakketten. Ze worden per functionaliteit besproken: algemeen gebruik, decimale nummers, kleur, tabulatie, tabellen over meerdere pagina's.
Toch is het moeilijk om door de bomen het bos te zien. De kleine beschrijving van de pakketten die aan het begin van het hoofdstuk wordt gegeven, dekt niet de volledige inhoud. De auteur geeft aan dat je zelf moet beslissen welke pakketten voor jezelf belangrijk of noodzakelijk zijn. En dat is nu net niet de reden waarom je zo een boek koopt: ofwel wil je als beginner bij de hand genomen worden en gewezen worden welke pakketten eruit springen of wil je als gevorderde snel een oplossing krijgen voor je probleem. De term 'algemeen gebruik' is bovendien te vaag om de soms heel probleemspecifieke pakketten in onder te brengen.
De beschrijving van de pakketten is overigens uitstekend. De uitleg is to the point, de opties worden goed beschreven en geïllustreerd met een voorbeeld. Er wordt geen woord verspeeld aan minder relevante info.
Herbert Voss weet uit omvangrijke pakketten de interessante gedeeltes met betrekking tot tabellen te halen. Zo haalt hij het uitgebreide pakket datatool aan om via een extern cvs-bestand data voor een tabel binnen te halen. Niet iets wat je direct verwacht maar het bewijst zijn oog voor detail en de kennis van de vele pakketten op CTAN. Dit geldt ook voor 'opeenvolgende' pakketten die vaak specifieke problemen van hun voorgangers weten op te lossen[4]. Het verschil wordt duidelijk in de tekst vermeld en geïllustreerd.
Wat ontbreekt in hoofdstuk 2 is een overzicht van de verschillende pakketten met hun eigenschappen en karakteristieken. Het is echter maar de vraag of dit met een dergelijke complexiteit überhaupt mogelijk is en een overzicht niet snel zou verzanden in details.

Werken met kleur en multi-pagina tabellen worden respectievelijk in hoofdstukken 3 en 4 aangepakt. In deze hoofdstukken word je bij de hand genomen om je doorheen de mogelijkheden en moeilijkheden van de diverse pakketten te leiden.

Waar hoofdstukken 1 tot 4 je 'zakelijke' informatie over de diverse pakketten aanbiedt, verandert de opzet in hoofdstukken 5 en 6. Hier komen 'tips and tricks' en concrete demo's die op mogelijkheden wijzen, aan bod. Dit tweeluik toont bovendien aan dat Voss voldoende op de hoogte is om LaTeX aan de man te brengen: kennis en praktijk smelten mooi samen en maken dit boek uitmuntend.

Sommige voorbeelden zijn niet alledaags: het gebruik van multicols in een tabel, Excel en Openofficebestanden gebruiken om tabellen aan te maken, het gebruik van PS-tricks om de layout te vervolmaken, tabellen roteren, automatische optellende nummers in elke nieuwe rij, enzovoort. Deze voorbeelden vind je weliswaar op diverse nieuwsgroepen en mailinglijsten terug maar ze zijn door de auteur geperfectioneerd en tot hun essentie teruggebracht zodat je snel tot een beter begrip en toepassing komt.

Eindconclusie is over het algemeen heel positief. Het boek zet je goed op weg om heel effectief met tabellen te werken op voorwaarde dat je het boek helemaal doorneemt. Bij een al te selectieve lezing kun je handige aanwijzingen en inzichten missen. De auteur slaagt er in om heel wat problemen te tackelen bij de werking 'binnen' in een tabel. Hij voorziet geen plaats om het principe van floats uit te leggen / te verklaren terwijl tabellen floating environments bij uitstek zijn. Misschien gaat hij er van uit dat zijn lezerspubliek dit reeds onder de knie heeft.

Kun je makkelijk overweg met de stijl van The LaTeX Companion dan ben je even goede vrienden met dit boek.

Typesetting tables with LaTeX , Herbert Voss, UIT Cambridge Ltd., 2011, 230 pagina's, ISBN 978-1-906860-25-7.

Hoofdstukken:

1. Introduction (pag. 3 - 17)
2. Packages (pag. 19 - 109)
3. Colour in tables (pag. 110 - 124)
4. Multi-page tables (pag. 125 - 166)
5. Tips and Tricks (pag. 167 - 176)
6. Examples (pag. 177 - 212)
7. Question and answers (pag. 213 - 216)

Noten

1. OpenOffice.org (`http://nl.openoffice.org`) of LibreOffice (`http://www.libreoffice.org`) zijn twee voorbeelden als ook Abiword (`http://www.abisource.com`).
2. The LaTeX Companion (Tools and Techniques for Computer Typesetting) door Frank Mittelbach, Michel Goossens, e.a. is een aanrader als naslagwerk voor elke beginnende en gevorderde LaTeX-gebruiker
3. Typesetting tables with LaTeX maar telt maar liefst 212 pagina's, index en inleiding niet meegerekend
4. Bijv. `tabulary` als aanvulling op `tabularx` om tabelbreedte aan te pakken.

Koen Wybo

Review of *Typesetting mathematics with LaTeX* by Herbert Voss

Introduction

In the Preface to the TeXBook [Knuth, 1986], Don Knuth wrote: "TeX [is] intended for the creation of beautiful books—and especially for books that contain a lot of mathematics." Some features of TeX have been adopted outside mathematics, for instance the hyphenation algorithm, but mathematical typesetting remains TeX's primary niche.

TeX is reputed to have a steep learning curve. It is quite different from other (alleged) typesetting systems, most of which have adopted the what-you-see-is-what-you-get model. The error messages confronting the beginner are often bewildering. As a guide for beginners, the TeXBook leaves a lot to be desired. However, the unrivaled quality of the resulting typeset output makes the effort to learn TeX worthwhile.

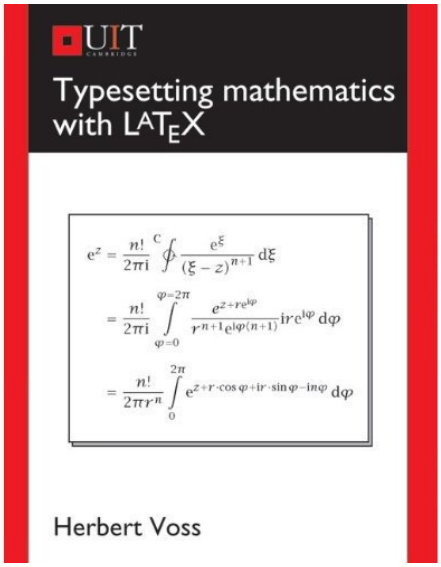
LaTeX [Lamport, 1994] was created by Leslie Lamport to make using TeX simpler, primarily by distinguishing the logical structure of a document from its typographical realization, and by providing a number of macros for constructions common in mathematical texts. LaTeX has now replaced pure TeX for most mathematical typesetting.

Most beginning LaTeX users in my experience (limited to computer science and physics departments in universities) start by modifying a LaTeX file from a colleague. Many of the high-level commands have well-chosen mnemonic names, and some users even add illuminating comments to their LaTeX files making the learning-by-modifying exercise much easier. However, inevitably there comes a time that this approach no longer works, and the dreaded stage of consulting a manual cannot be avoided. It seems that Voss' book is aimed at users in that situation.

Description

This book is a translation of the German version [Voß, 2009]. It is similarly priced, so you can choose either based on your linguistic preferences.

The book has eleven chapters. After the (one-page) Introduction, chapter 2 treats in-line mathematical expressions, using standard LaTeX. Chapter 3 treats display mode mathematics, while chapter 4 describes other constructs standard LaTeX offers for mathematical



formulae. Compared to Lamport's book, these chapters contain little that is new, either in substance or in presentation. What is new, is sometimes bordering on the esoteric, for example the discussion of the interchanged role of comma and point in (American) English versus 'continental' usage, and the repercussions this has for spacing large numbers. Doubtless this difference should be hidden in a macro.

Chapter 5 gives a brief, but nevertheless too long, description of the `color` package to colour formulae. Chapter 6 describes the various packages provided by the American Mathematical Society (without reference to the admittedly old manual by M.D. Spivak [Spivak, 1986]). Chapter 7, with 50 pages the longest chapter, lists mathematical symbols provided by standard LaTeX as well as a number of other packages. As the author says: "The order of the packages in this chapter is purely based on optimising page breaks". This does not really help in guiding the user to find the package that may contain the symbol he/she is looking for.

Chapter 8 describes some of the internals of TeX, while chapter 9 lists a number of packages with material useful for typesetting mathematical formulae, and gives some examples of using them. Again in the author's

words: "The selection in this chapter is more or less arbitrary, but common problems are addressed." Chapter 10 gives a number of examples of various constructions, without a clear organization, based 'on personal experience'. Finally, chapter 11 discusses which text and math fonts can be combined with good typographical results; this is illustrated with examples.

Evaluation

It is quite unclear to me what audience the author had in mind when writing this book. Newcomers will find the book impenetrable: it assumes prior knowledge of concepts like packages, character codes, and macro writing. To illustrate: without elaborating further, on page 24 the author suggests "Remember when using this package [...] that you have to embed this reset command in a `makeatletter ...makeatother` sequence though."

Advanced users, who are familiar with these concepts, may wish to use this book to find solutions to the question "How to code this particular mathematical formula in LaTeX?" If they do, they encounter a problem: the lack of organization of the book. Although the title mentions typesetting mathematics, the organization is not based on mathematical constructs at all (unless by accident, because most packages described *are* for specialized mathematical constructs). The index does not help here: if one does not yet know the name of the macro to use, exhaustive search (of the book, not of the index!) seems the only way to find it.

Organization aside: one expects a book on typesetting mathematics to be meticulously copy-edited. Unfortunately, that is not the case here. Already in the Preface (p. vii), the reader is confronted with an unresolved page reference **(??)**; there are several more of these. In the (brief) index of persons, N.B. Taylor can be found under *N* rather than *T*. The cover uses American spelling, while the book uses British English.

Surprisingly, the bibliography does not include Lamport's book [Lamport, 1994]. Both Lamport and Knuth are absent from the index of persons.

In summary, it is not clear what category of users can benefit from this book. There is certainly a need for a book that helps beginning users, versed in mathematics but not in typographical tools, to convert their ideas into beautifully typeset documents. This is not that book.

Typesetting mathematics with LaTeX , Herbert Voss, UIT Cambridge Ltd., 2010, 304 pagina's, ISBN 978-1-906860-17-2.

References

[Knuth, 1986] Knuth, D. E. (1986). *The TeXbook*. Addison-Wesley Publishing Company.

[Lamport, 1994] Lamport, L. (1994). *LaTeX a Document Preparation System*. Addison-Wesley Publishing Company, 2nd edition.

[Spivak, 1986] Spivak, M. D. (1986). The joy of TeX. A gourmet guide to typesetting with the AmSTeX macropackage. American Mathematical Society.

[Voß, 2009] Voß, H. (2009). Mathematiksatz mit LaTeX. Lehmanns Media-Lob.de.

Nicolaas J.I. Mars
University of Groningen, Groningen, The Netherlands
n.j.i.mars@alumnus.utwente.nl

A Personal Organizer: PocketDiary

(a module)

Abstract
Sometimes, a cheap personal organizer on paper can come in handy. This solution prepared in ConTeXt MKIV provides a range of options to set up such a personal organizer. The point is, that the PocketDiary is printed on a single sided A4 landscape sheet of paper and then folded into a pocket size booklet hereby preventing that unprinted/empty pages are seen. The PocketDiary is easy to make and after 1 week it is simply replaced with a subsequent booklet. A detailed description is given of the system and how to set up a production file. At the end of the article instructions are included how to fold the booklet.

Keywords
Maps, Context, module, lua

Introduction

Some time ago my brother Heinz asked me to prepare him a special page-arrangement scheme. This scheme is suitable to form a section with a single-sided printed sheet of paper. He wanted to use it for a special kind of greeting-cards. By coincidence I detected an article by U. Ziegenhagen in de TEXnische Kommödie nr. 3/2010 [4]. This article deals with the preparation of a PocketMod, which is a personal organizer based on the aforementioned arranging scheme. PocketMod is available as an online version on http://www.pocketmod.com. – After reading the article and visiting the web-site I got intrigued by the fact that ConTEXt has built-in arranging capabilities and due to the LuaTEX engine it should be possible to build such a personal organizer in ConTEXt.

The result is a module which can generate such personal organizers, called PocketDiary. I give a description of the possibilities in this article.

General Description

The PocketDiary is a personal organizer. It is like a section of a book, but very small in size. The PocketDiary is based on the idea, that it should be possible to repeatedly produce such a personal organizer during the year by altering a minimal number of variables. The diary is based on a week number and, of course, the year. – Although this is enough information for preparing a week-diary, it is not enough if you want more options such as inclusion of month-tables, a year-table, inclusion of personal data etc. To start up the PocketDiary production more information needs to be given to the system.

PocketDiary Layout

The code presented hereunder is all you need to put into your personal PocketDiary file to be able to produce a personal organizer.

First we program ConTEXt to load the module *t-pocketdiary*

```
\usemodule[pocketdiary]
```

Without stating the font to be used PocketDiary uses Latin Modern. If you have your own favourite font set it up here:

```
\usetypescriptfile[type-seravek]
\usetypescript[Seravek]
\setupbodyfont[Seravek,ss,9pt]
```

PocketDiary has a multiple-language interface, so we need to tell it which language we use. Therefore the document should start with the definition of the main language used. Supported interfaces are English, Dutch, German, Italian and French.

```
\mainlanguage[nl]
```

The module automatically sets the layout of the PocketDiary page.

The PocketDiary's page design is based on a header and footer line and with the calendar information in the main text area. The header line will contain main information on the calendar-issue at hand i.e. Day of the month – Day name (short) – Week number – Year.

Everything which is configurable, is brought together in 5 sets of variables.

In order to be able to calculate the desired calendars, the following variables of PocketDiary should be set:

```
\setvariables
   [PocketDiary]
   [Year=2011,
    Week=17,
    Day=7,
    Month=5,
    Nextyear=yes]
```

The variables in the set 'PocketDiary' are explained in the following table. Be aware, that some of the variables change the behaviour of the PocketDiary.

Variable	Value	Comment
Year	number	Year numbers in the range 1900 and 4099. The lower limit is computer dependent (OS-timestamp), the upper limit is dependent on the Easter Sunday calculation.
Week	number	Values between 1 and 53.
Day	number	Values between 1 and 7. If this variable contains a value, then the PocketDiary will be made up according to the variable specifications given in the 'PocketDiaryLayout' section. If this variable is empty, then a PocketDiary with one page per day is made up. The content of pages 1 and 8 can be chosen freely in the section 'PocketDiaryLayout'.
Month	number	Values between 1 and 12. This variable is independent from the variables 'Year' and 'Week'. It indicates only which month should be calculated for a month table.
Nextyear	yes/no	The testing is done on 'yes'. If set to 'yes' the next year instead of the current year is used for the calculation of the year calendar. This can be practical when current year is nearly over.

The PocketDiary holds 8 (Page1 to Page8) pages. It is possible to give the organizer an individual layout according to your own wishes. It is no problem to place several pages with the same template name.

```
\setvariables
   [PocketDiaryLayout]
```

```
   [Page1=Lost-Return,
    Page2=Week,
    Page3=Day,
    Page4=Monthcurrent,
    Page5=Blank,
    Page6=Contact,
    Page7=Caro,
    Page8=Lines]
```

Variable	Comment
Day	The weekday, indicated in the variable 'Day' in the previous section, is used to make a PocketDiary page.
Week	A week calendar, based on the variable 'Week' in the previous section, is used for the presentation of a week table.
Monthcurrent	A month table based on the value in the variable 'Month' of the previous section, is typeset.
Monthnext	A month table of the next month based on the value in the variable 'Month' of the previous section is typeset.
Yearcalendar	A complete year calendar is typeset. If the 'Nextyear' variable is not 'yes', the year calendar for the year indicated in variable 'Year' of the previous section is used. Otherwise that variable is increased by 1 to typeset the year calendar of the following year.
Lost-Return	By means of the set values in the section 'PocketDiaryAddress' a lost-and-return page is composed.
Blank	This page carries a header and a footer but is otherwise empty.
Todo	A todo-list template is typeset.
Caro	A page with full-grid-paper is typeset.
Lines	A page with grid lines is typeset
Contact	A form with two sets of preprinted fields for marking down contact information is typeset.

The third block of variables contains information used for the footer and the lost-return form.

```
\setvariables
   [PocketDiaryAddress]
   [Familyname=Egger,
    Forename=Willi,
    Street=Townstreet 3B,
    Zipcode=5000,
    City=New CONTEXT,
    Country=TEX-world,
    Phone=+22 444 55 88 66,
    Mobile=+22 6 19 19 1717,
    E-mail=info at pocketdiary.org,
    Web=www.pocketdiary.org]
```

The footer line contains three fields which can be customized with the 'PocketDiaryFooter' variables.

```
\setvariables
   [PocketDiaryFooter]
   [Lefttext=PocketDiary,
    Centertext=\pagenumber,
```

```
Righttext={\getvariable{PocketDiaryAddress}{Forename},~{\currentdate[year]}}]
```

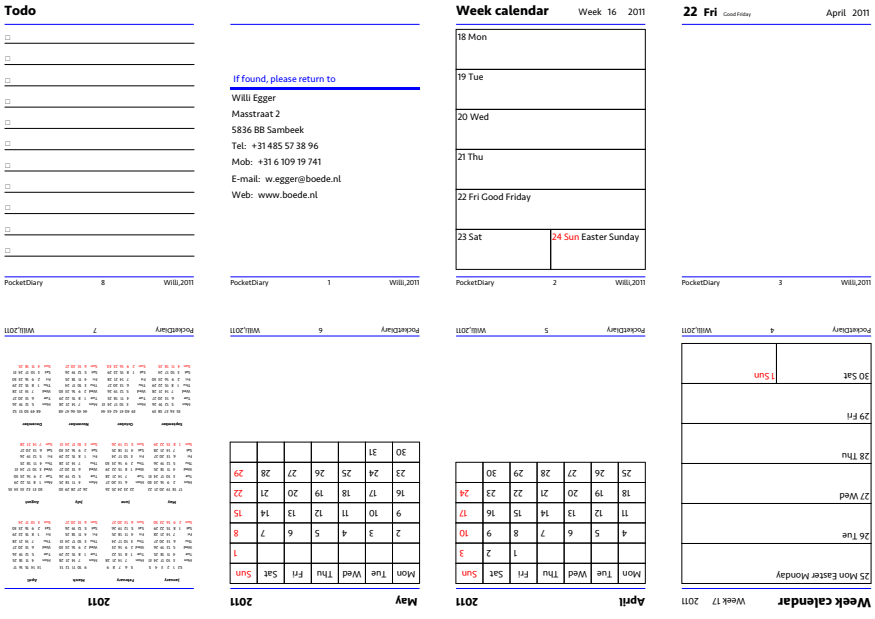
The main text is separated from the header and footer with a line. This line can be given a custom color. The standard color is blue. For those who wish grid lines in another color than light-gray, they can give ‘Gridline’ another value.

```
\setvariables
  [PocketDiaryColors]
  [Separatorline=blue,
   Gridline={s=.75}]
```

The last block of code states:

```
\starttext
       \setups{PocketDiary}
\stoptext
```

The result: Day-Calendar with a Year Calendar



The result: Week Calendar with Two Custom Pages

Available templates

Day. The day calendar is shown in figure 1.
The weekend calendar shows Saturday and Sunday on one page (see figure 2)

Week Calendar. The week calendar looks as shown in the example figure 3.

Month Calendar. The month calendar is shown in figure 4.

Year Calendar. An example of the current year calendar you can find in figure 5.

Templates Unrelated to Calendar Calculations. The PocketDiary comes with a couple of templates for writing down information. They are shown in figure 6.

Figure 1. The day calendar

Figure 2. The weekend calendar

Figure 3. The week calendar

Figure 4. The month calendar

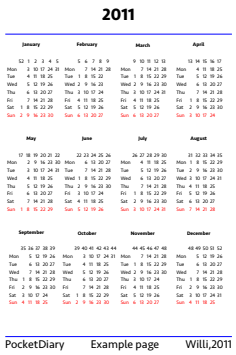


Figure 5. The year calendar

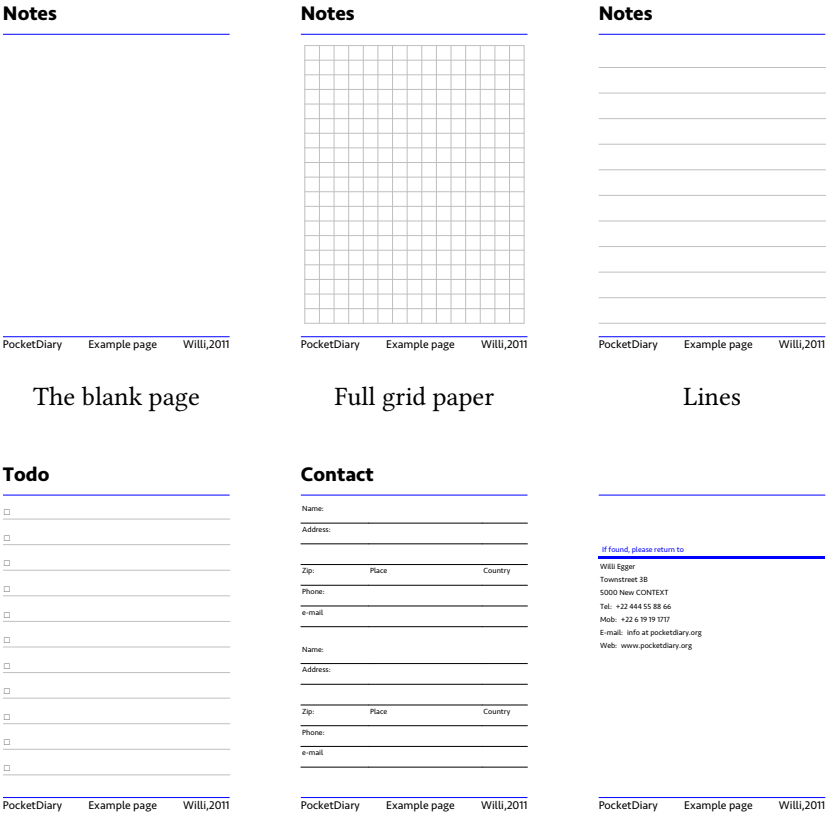


Figure 6. The templates of PocketDiary

The Lua Part

First I thought that it would not be that hard to build a calendar for this kind of personal organizer. It is fairly easy, if you expect just a basic organizer without fancy bells and whistles. However one should not be scared to type again and again a lot of information in order to get the desired output. – So, when I started this, I wanted to

get a production file, which would be easy to maintain and there should not be too many changes during repeated cycles of making such a personal organizer. This implies however, that the background machinery must be able to calculate everything based on a year, week number, month and a day number. – Soon I detected that there are more and less difficult areas in the year's calendar. The easier things comprise leap-year calculations and calculations which do not involve january and december. The beginning of the year and the end of the year are quite tricky moments. Think e.g. whether the last days of december are in week 52 or 53 and whether the first days of the next year belong to week 52, 53 or are representing the first week of the new year.

The decision has been made, that this module would use the European standard for date calculations. The Gregorian calendar is used exclusively. The module is based on the ISO-8061 standard, which defines that the first week of a year is the week having the year's first Thursday in it. Furthermore it states that the week starts on Monday which is different from what Lua does, where day 1 is Sunday. The os-library coming with lua can do a lot for date calculations. So, where possible this library is used.

Googling helps a lot to gain insight in the date-calculations. The work of Sohael Babwani [5] is an interesting source. I wanted the PocketDiary to indicate also the main Christian feast days. I found the web-site of Ron Mallen [6], which was a big help. He developed an Easter Sunday calculation based on three tables. He also presents a BASIC programme on his web-site which calculates the date automatically. This programme is then transcribed into Lua.

There are more questions to be answered e.g. how to calculate a week number from a certain date. Thanks to Ferry van Schaik [7] I got a Java snippet, which I have converted to Lua.

How to Fold the PocketDiary

The eight printed pages are folded in such a way that the PocketDiary presents itself as a small booklet. There are no empty pages visible.

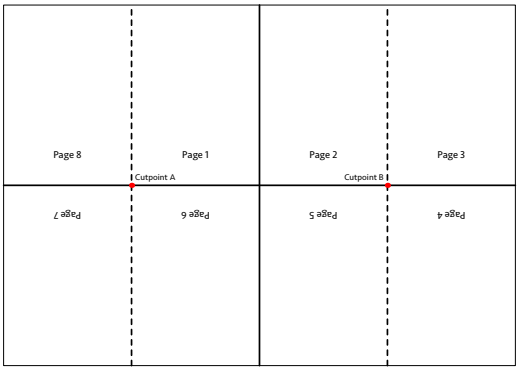


Figure 7. The basic folding scheme

First make two mountain-folds as indicated with the straight lines in figure 7. Unfold the paper and turn it face up and 90° to the left. Make a valley-fold with the lower part of the sheet onto the previously made mountain-fold. Unfold and turn the sheet 180°. Make another valley-fold as described before. Unfold the sheet.

Take a sharp knife and a ruler. Cut the paper open between cutting point A and B (see figure 7).

Now we can fold the booklet. First, fold the paper again lengthwise. Then hold the double folded paper with the mountain-fold up. Push from both sides towards

the center in order to get a form similar to figure 8. Then fold the upper double-page in direction B, the lower double-page in direction C and finally the lefthand double-sided page in direction D.

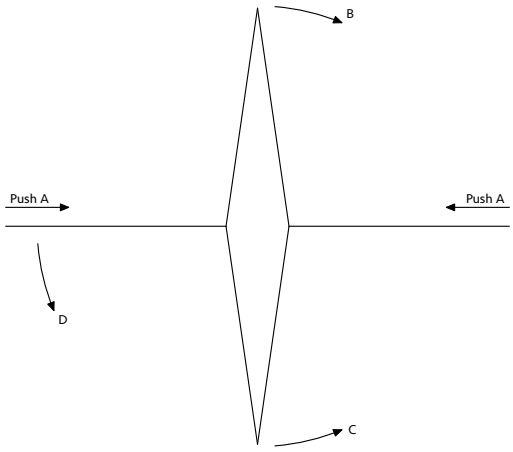


Figure 8. The basic folding scheme

Before creasing the booklet at the spine it is worthwhile to put the section down on the table and adjust folds where needed. Finally the spine is creased preferably with a bone-folder.

The Module

The module is available on the ConTEXt wiki and it can be downloaded from http://modules.contextgarden.net/.

References

1. Programming in Lua. Roberto Ierusalimschy. Lua.org, Rio de Janeiro. 2nd ed. 2006.
2. Metafun. Hans Hagen. PRAGMA ade, Hasselt. Version 2. 2010.
3. ConTEXt Lua documents. Hans Hagen. PRAGMA ade, Hasselt. prelim. version. 2010.
4. Pocketmods mit LATEX erstellen. Uwe Ziegenhagen. In TEXnische Kommödie 3/2010, Dante e.V, Heidelberg. pp. 27-32. 2010.
5. An extended Approach to the Julian and Gregorian Calendar. Sohael Babwani. 2010. From:http://www.babwani-congruence.blogspot.com/
6. Easter Dating Method. Ron W. Mallen. Astronomical Society of Southern Australia. 2002. Web-site: http://www.assa.org.au/edm.html.
7. How can I calculate the week number of the current date?. Ferry van Schaik. 2001. From:http://www.irt.org/script/914.htm

Thanks

I would like to thank Hans Hagen, Taco Hoekwater and Hartmut Henkel for the great LuaTEX machinery and Wolfgang Schuster for supporting me in tackling the multi-lingual interface. Silvan, thank you for proofreading this text.

Willi Egger
w.egger@boede.nl

Tagged PDF

Introduction

Occasionally users asked me if ConTEXt can produce tagged pdf and the answer to that has been: I'll implement it when I need it. However, users tell me that publishers show an increasing demand for tagged pdf files, although one might wonder what for, except maybe for accessibility. Another reason for not having spent too much time on it before, is that the specification was not that inviting.

At any rate, when I saw Ross Moore[1] presenting tagged math at TUG 2010, I decided to look up the spec once more and see if I could get into the mood to implement tagging. Before I started it was already clear that there were a couple of boundary conditions:

- Tagging should not put a burden on the user but users should be able to do the tagging themselves.
- Tagging should not slow down a run too much; this is no big deal as one can postpone tagging till the last run.
- Tagging should in no way interfere with typesetting, so no funny nodes should be injected.
- Tagging should not make the code look worse, neither the document source, nor the low level ConTEXt code.

And of course implementing it should not take more than a few days' work, certainly not during an exceptionally hot summer.

You can 'google' for one of Ross's documents (like `DML_002-2009-1_12.pdf`) to see how a document source looks at his end using a special version of pdfTEX. However, the version on my machine didn't support the primitives shown, so I could not see what was happening under the hood. Unfortunately it is quite hard to find a properly tagged document so we have only the reference manual as starting point. As the pdfTEX approach didn't look that pleasing anyway, I just started from scratch.

Tags can help Acrobat Reader when reading out the text aloud. But you cannot browse the structure in the no-cost version of Acrobat and as not all users have the professional version of Acrobat, the fact that a document has structure can go unnoticed. Add to that the fact that the overhead in terms of bytes is quite large as many more objects are generated, and you will understand why this feature is not enabled by default.

Implementation

So, what does tagging boil down to? We can best look at how tagged information is shown in Acrobat. Figure 1 shows the content tree that has been added (automatically) to a document while figure 2 shows a different view.

In order to get that far, we have to do the following:

- Carry information with (typeset) text.
- Analyse this information when shipping out pages.

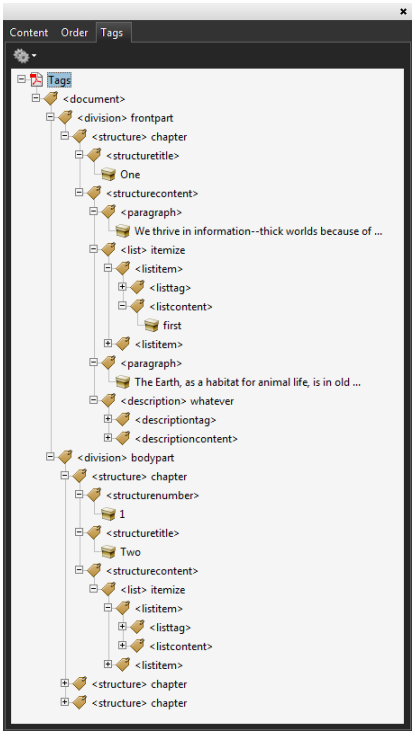


Figure 1. A tag list in Acrobat.

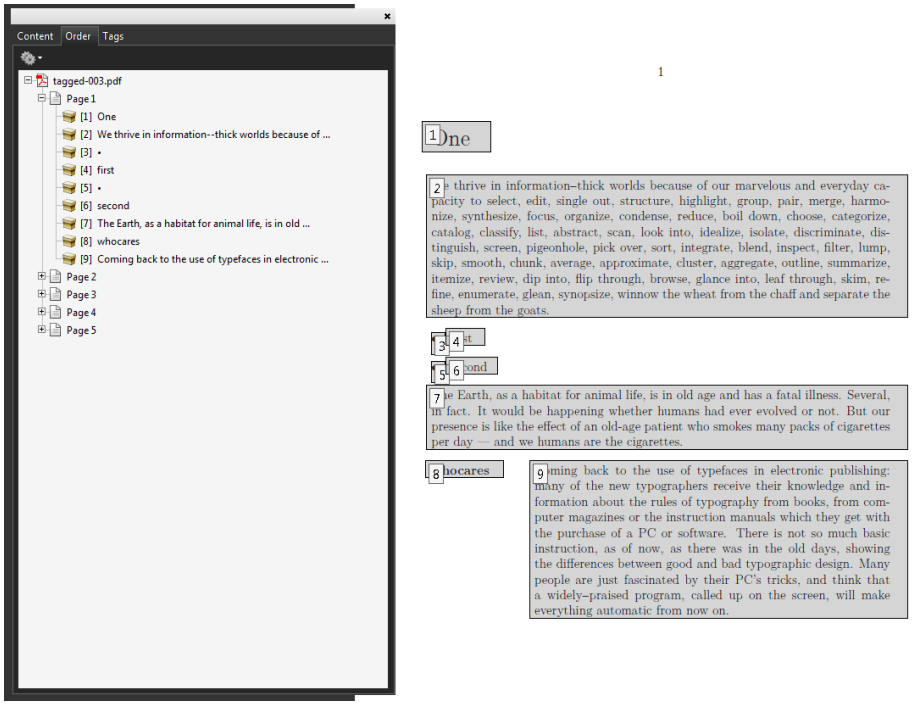


Figure 2. Acrobat showing the tag order.

- Add a structure tree to the page.
- Add relevant information to the document.

That first activity is rather independent of the other three and we can use that information for other purposes as well, like identifying where we are in the document. We carry the information around using attributes. The last three activities took a bit of experimenting mostly using the "Example of Logical Structure" from the pdf standard 32000-1:2008.

This resulted in a tagging framework that uses explicit tags, meaning the user is responsible for the tagging:

```
\setupstructure[state=start,method=none]

\starttext

\startelement[document]

    \startelement[chapter]
        \startelement[p] \input davis \stopelement \par
    \stopelement

    \startelement[chapter]
        \startelement[p] \input zapf \stopelement \par
        \startelement[whatever]
            \startelement[p] \input tufte \stopelement \par
            \startelement[p] \input knuth \stopelement \par
        \stopelement
    \stopelement

    \startelement[chapter]
        oeps
        \startelement[p] \input ward \stopelement \par
    \stopelement

\stopelement

\stoptext
```

Since this is not much fun, we also provide an automated variant. In the previous example we explicitly turned off automated tagging by setting `method` to `none`. By default it has the value `auto`.

```
\setupstructure[state=start] % default is method=auto

\definedescription[whatever]

\starttext

\startfrontmatter
    \startchapter[title=One]
        \startparagraph \input tufte \stopparagraph
        \startitemize
            \startitem first \stopitem
            \startitem second \stopitem
        \stopitemize
        \startparagraph \input ward \stopparagraph
        \startwhatever {Herman Zapf} \input zapf \stopwhatever
    \stopchapter

\stopfrontmatter
```

```
\startbodymatter
    ..................
```

If you use commands like `\chapter` you will not get the desired results. Of course these can be supported but there is no real reason for it, as in MkIV we advise using the `start-stop` variant.

It will be clear that this kind of automated tagging brings with it a couple of extra commands deep down in ConTEXt and there (of course) we use symbolic names for tags, so that one can overload the built-in mapping.

```
\setuptaglabeltext[en][document=text]
```

As with other features inspired by viewer functionality, the implementation of tagging is independent of the backend. For instance, we can tag a document and access the tagging information at the TEX end. The backend driver code maps tags to relevant pdf constructs. First of all, we just map the tags used at the ConTEXt end onto themselves. But, as validators expect certain names, we use the pdf rolemap feature to map them to (less interesting) names. The next list shows the currently used internal names, with the pdf ones between parentheses.

construct (Span), delimited (Quote), delimitedblock (BlockQuote), description (Div), descriptioncontent (Div), descriptionsymbol (Span), descriptiontag (Div), division (Div), document (Div), float (Div), floatcaption (Caption), floatcontent (P), floattag (Span), floattext (Span), formula (Div), formulacontent (P), formulaset (Div), formulatag (Span), image (P), item (Li), itemcontent (LBody), itemgroup (L), itemtag (Lbl), link (Link), list (TOC), listcontent (P), listdata (P), listitem (TOCI), listpage (Reference), listtag (Lbl), margintext (Span), margintextblock (Span), math (Div), merror (Span), mfrac (Span), mi (Span), mn (Span), mo (Span), mover (Span), mpgraphic (P), mroot (Span), mrow (Span), ms (Span), msqrt (Span), msub (Span), msubsup (Span), msup (Span), mtext (Span), munder (Span), munderover (Span), paragraph (P), register (Div), registerentries (Div), registerentry (Span), registerpage (Span), registerpagerange (Span), registerpages (Span), registersection (Div), registersee (Span), registertag (Span), section (Sect), sectioncontent (Div), sectionnumber (H), sectiontitle (H), sort (Span), subformula (Div), subsentence (Span), synonym (Span), table (Table), tablecell (TD), tablerow (TR), tabulate (Table), tabulatecell (TD), tabulaterow (TR), verbatim (Code), verbatimblock (Code), verbatimline (Code), verbatimlines (Code).

So, the internal ones show up in the tag trees as shown in the examples but applications might use the rolemap which normally has less detail.

Since we keep track of where we are, we can also use that information for making decisions.

```
\doifinelementelse{structure:section}            {yes} {no}
\doifinelementelse{structure:chapter}            {yes} {no}
\doifinelementelse{division:*-structure:chapter} {yes} {no}
\doifinelementelse{division:*-structure:*}       {yes} {no}
```

As shown, you can use `*` as a wildcard. The elements are separated by `-`. If you don't know what tags are used, you can always enable the tag related tracker:

```
\enabletrackers[structure.tags]
```

This tracker reports the identified element chains to the console and log.

Special care

Of course there are a few complications. First of all the tagging model sort of contradicts the concept of a nicely typeset document where structure and outcome are not always related. Most TEX users are aware of the fact that TEX does not have space characters and does a great job on kerning and hyphenation. The tagging machinery on the other hand uses a rather dumb model of strings separated by spaces.[2] But we can trick TEX into providing the right information to the backend so that words get nicely separated. The non-optimized function that does this looks as follows:

```
function injectspaces(head)
    local p
    for n in node.traverse(head) do
        local id = n.id
        if id == node.id("glue") then
            if p and p.id == node.id("glyph") then
                local g = node.copy(p)
                local s = node.copy(n.spec)
                g.char, n.spec = 32, s
                p.next, g.prev = g, p
                g.next, n.prev = n, g
                s.width = s.width - g.width
            end
        elseif id == node.id("hlist") or id == node.id("vlist") then
            injectspaces(n.list,attribute)
        end
        p = n
    end
end
```

Here we squeeze in a space (given that it is in the font which it normally is when you use ConTEXt) and make a compensation in the glue. Given that your page sits in box 255, you can do this just before shipping the page out:

```
injectspaces(tex.box[255].list)
```

Then there are the so-called suspects: things on the page that are not related to structure at all. One is supposed to tag these in a special way to prevent the built-in reading equipment from getting confused. So far we could get around them simply because they don't get tagged at all and therefore are not seen anyway. This might well be enough of a precaution.

Of course we need to deal with mathematics. Fortunately the presentation MathML model is rather close to TEX and so we can map onto that. After all we don't need to care too much about back-mapping here. The currently present code is rather experimental and might get extended or thrown out in favour of inline MathML. Figure 3 demonstrates that a first approach does not even look that bad. In future versions we might deal with table-like math constructs, like matrices.

This is a typical case where more energy has to be spent on driving the voice of Acrobat but I will do that when we find a good reason.

As mentioned, it will take a while before all relevant constructs in ConTEXt support tagging, but support is already quite complete. Some screen dumps are included as examples at the end.

Figure 3. Experimental math tagging.

Conclusion

Surprisingly, implementing all this didn't take that much work. Of course detailed automated structure support from the complete ConTEXt kernel will take some time to get completed, but that will be done on demand and when we run into missing bits and pieces. It's still not decided to what extent alternate representations and alternate texts will be supported. Experiments with the reading-aloud machinery are not satisfying yet but maybe it just can't get any better. It would be nice if we could get some tags being announced without overloading the content, that is: without using ugly hacks.

And of course, code like this is never really finished if only because pdf evolves. Also, it is yet another nice test case and torture test for LuaTEX and it helps us to find buglets and oversights.

Some more examples

In ConTEXt we have user definable verbatim environments. As with other user definable environments we show the specific instance as comment next to the structure component. See figure 4. Some examples of tables are shown in figure 5. Future versions will have a bit more structure. Tables of contents (see figure 6) and registers (see figure 7) are also tagged. (One might wonder how useful this is.) In figure 8 we see some examples of floats. External images as well as MetaPost graphics are tagged as such. This example also shows an example of a user environment, in this case:

```
\definestartstop[notabene][style=\bf]
```

In a similar fashion, footnotes (figure 9) end up in the structure tree, but in the typeset document they move around (normally forward when there is no room).

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

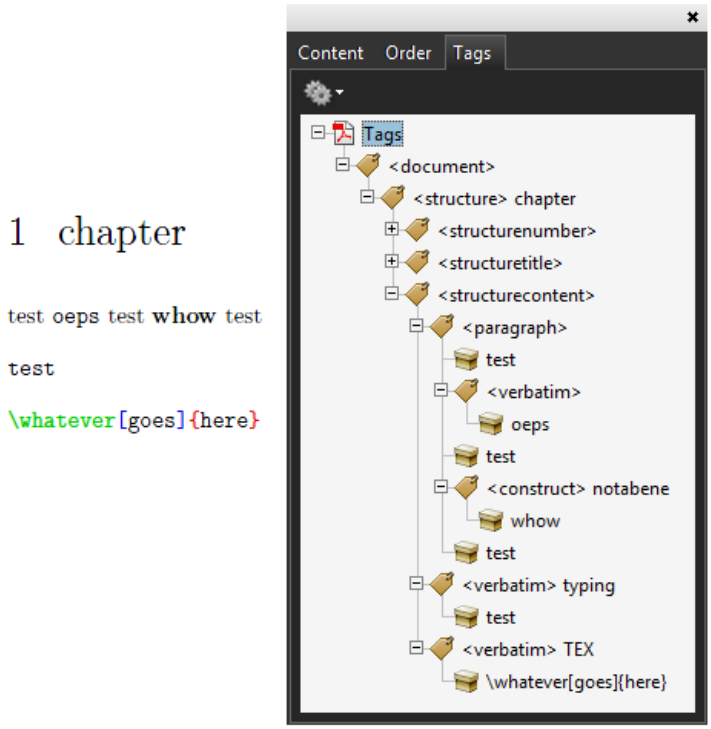


Figure 4. Verbatim, including dedicated instances.

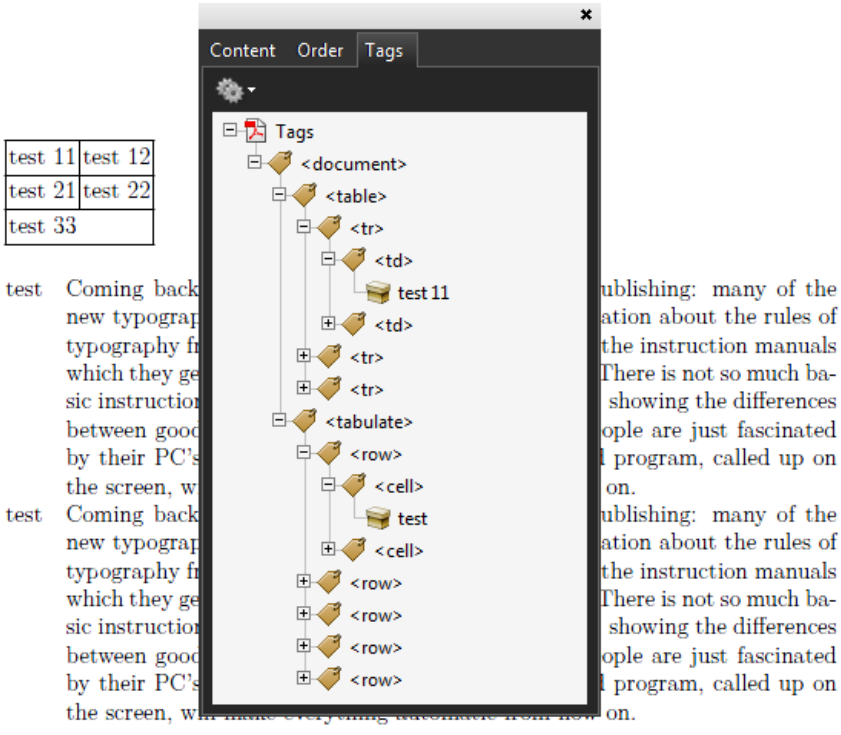


Figure 5. Natural tables as well as the tabulate mechanism is supported.

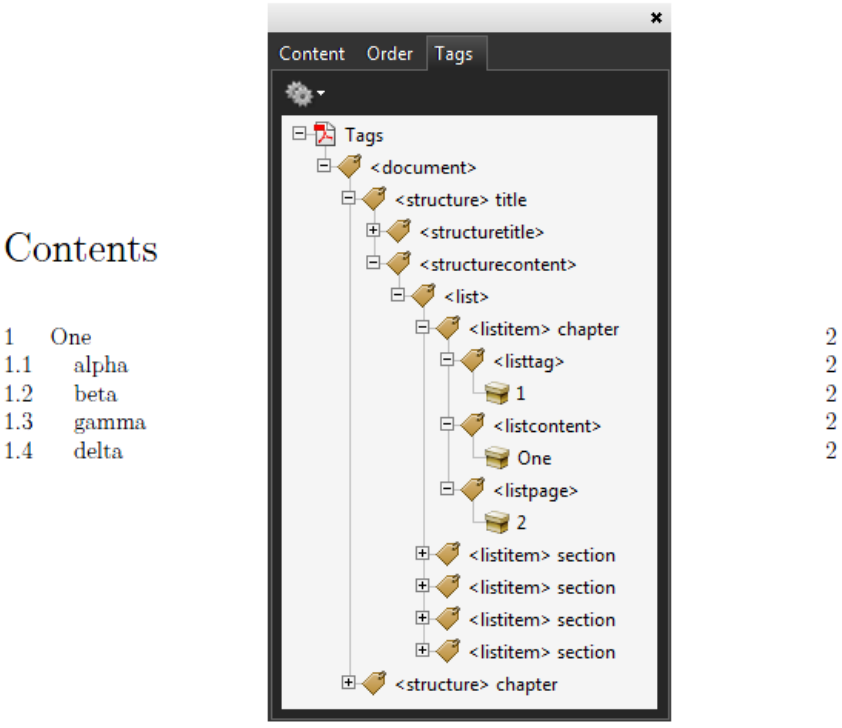


Figure 6. Tables of content with specific entries tagged.

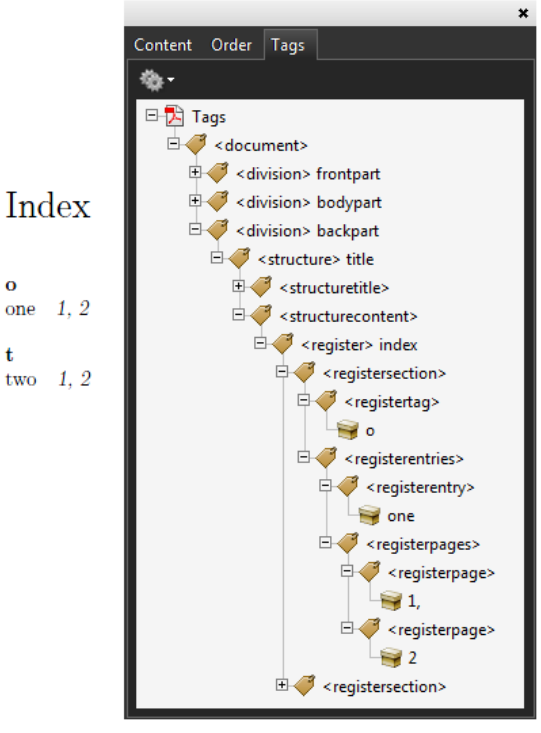


Figure 7. A detailed view of registered is provided.

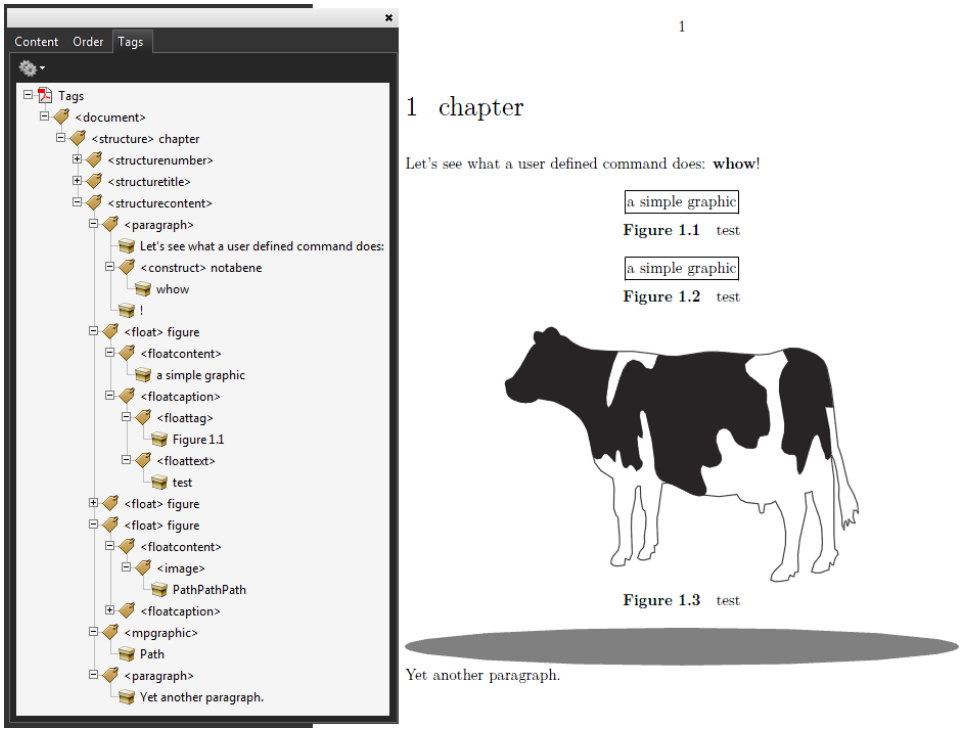


Figure 8. Floats tags end up in text stream. Watch the user defined construct.

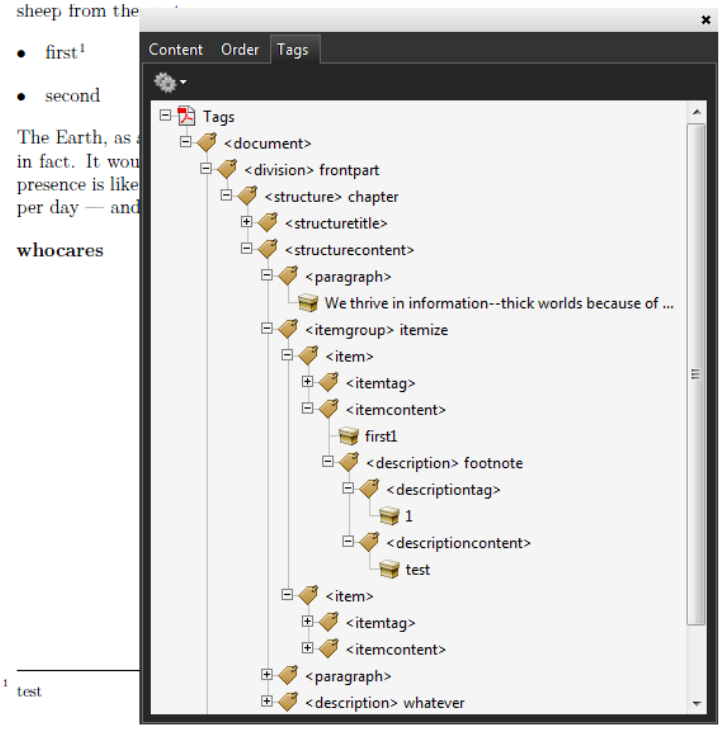


Figure 9. Footnotes are shown at the place in the input (flow).

Inter-character spacing and ligatures[1]

There was a discussion on the LuaTEX (dev) list about inter character spacing and ligatures. The discussion involved a mechanism inherited from pdfTEX but in ConTEXt we don't use that at all. Actually, support for inter character spacing was added in an early stage of MkIV development as an alternative for the MkII variant, which used parsing at the TEX end. Personally I never use this spacing, unless a design in a project demands it.

In the MkIV method we split ligatures when its components are known. This works quite well. It's anyway a good idea to disable ligatures, so it's more of a fallback. Actually we should create components for hard coded characters like æ but as no one ever complained I leave that for a later moment.

As we already had the mechanisms in place, support for selective spacing of ligatures was a rather trivial extension. If there is ever a real need for it, I will provide control via the normal user interface, but for now using a few hooks will do. The following code shows an example of an implementation.

```
local utfbyte = utf.byte

local keep = {
    [0x0132] = true, [0x0133] = true, -- IJ ij
    [0x00C6] = true, [0x00E6] = true, -- AE ae
    [0x0152] = true, [0x0153] = true, -- OE oe
}

function typesetters.kerns.keepligature(n)
    return keep[n.char]
end

local together = {
    [utfbyte("c")] = { [utfbyte("k")] = true },
    [utfbyte("i")] = { [utfbyte("j")] = true },
    [utfbyte("I")] = { [utfbyte("J")] = true },
}

function typesetters.kerns.keeptogether(n1,n2)
    local k = together[n1.char]
    return k and k[n2.char]
end
```

The following also works:

```
local lpegmatch = lpeg.match
local fontdata  = fonts.identifiers

local keep = -- start of name
        lpeg.P("i_j")
      + lpeg.P("I_J")
      + lpeg.P("aeligature")
```

1. Excerpt from the 'Weird Examples' chapter in hybrid.pdf

```
      + lpeg.P("AEligature")
      + lpeg.P("oeligature")
      + lpeg.P("OEligature")

function typesetters.kerns.keepligature(n)
    local d = fontdata[n.font].descriptions
    local c = d and d[n.char]
    local n = c and c.name
    return n and lpegmatch(keep,n)
end
```

A more generic solution would be to use the `tounicode` information, but it would be overkill as we're dealing with a rather predictable set of characters that have gotten Unicode slots assigned. When using basemode most fonts will work anyway.

So, is this really worth the effort? Take a look at the following example.

```
\definecharacterkerning [KernMe] [factor=0.25]

\start
  \setcharacterkerning[KernMe]
  \definedfont[Serif*default]
  Ach kijk effe, \ae sop draagt een knickerbocker! \par
  \definedfont[Serif*smallcaps]
  Ach kijk effe, \ae sop draagt een knickerbocker! \par
\stop
```

Typeset this (Dutch text) looks like:

Ac h k ij k e f f e, æ s o p d r a a g t ee n k n i cke r bocke r !
Ac h k ij k e f f e, æ s o p d r a a g t ee n k n i cke r bocke r !

You might wonder why I decided to look into it. Right at the moment when it was discussed, I was implementing a style that needed the Calibri font that comes with MS Windows, and I visited the FontShop website to have a look at the font. To my surprise it had quite some ligatures, way more than one would expect.

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

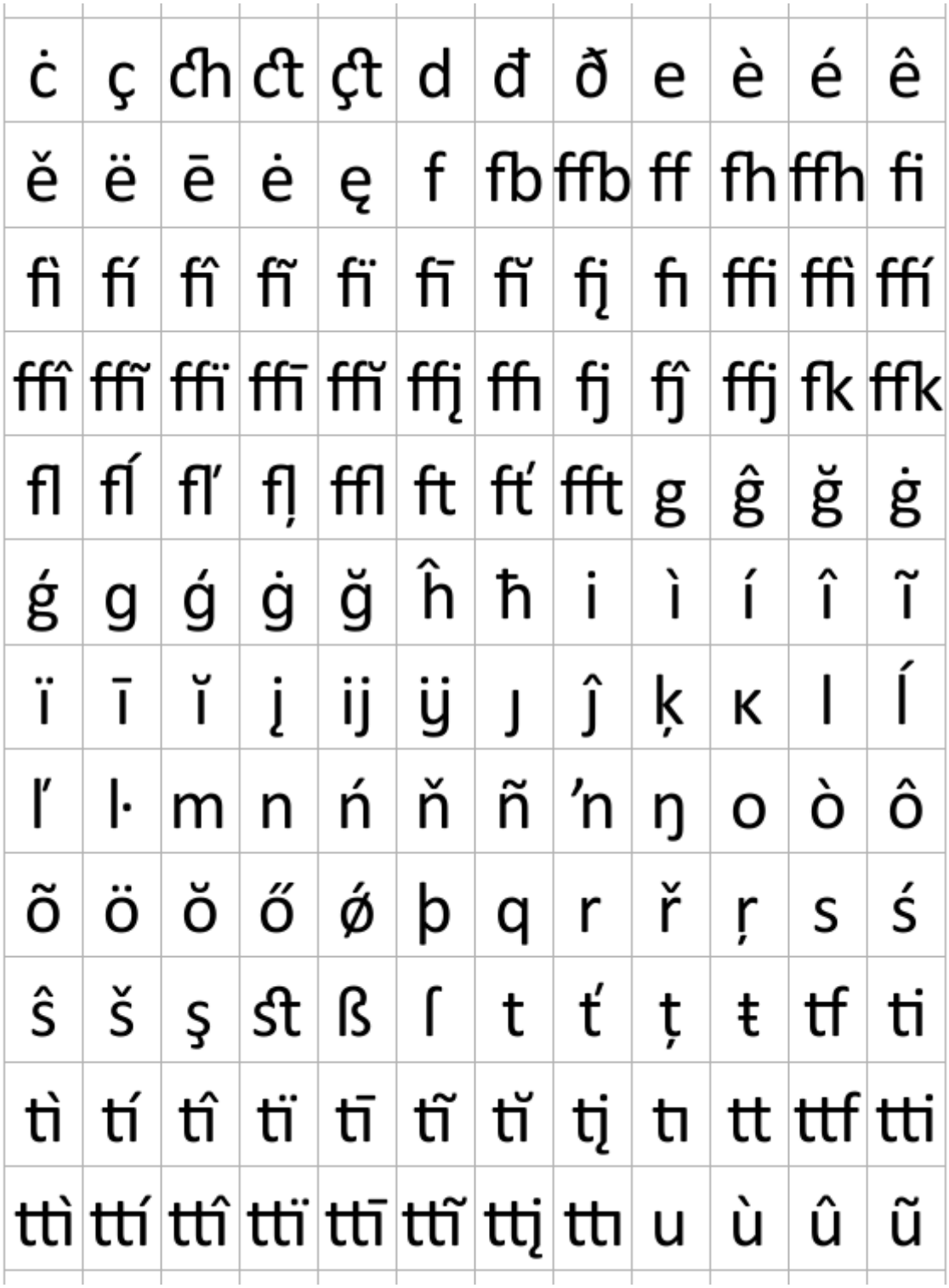


Figure 1. Some of the ligatures in Calibri Regular. Just wonder what intercharacter spacing will do here.

8th March

—an OTF exercise of Cyrillic by PostScript—

Abstract
An OTF with Cyrillic — keyboard and glyphs — is used in PostScript for an 8th March congratulation. The wired-in ASCII code table in TEX inhibits keyboarding Cyrillic.

Keywords
Adobe, afii, ASCII, CID, Cyrillic, EPSF, encoding vector, keyboard layout, lemniscate, minimal encapsulated PostScript, OTF, plain TeX, Photoshop, PSlib, qwerty, TEXworks

Introduction

In the beginning of March I saw an advertisement on Russian (satellite) tv with a congratulation along the path of the digit 8 on occasion of the international women day. This inspired me to make a present for my wife: a lemniscate as path with congratulations in Russian along its path by PostScript.

On the internet I found many congratulations as YTUBE videos, only a few photographs

20 years ago I might have tried to use TEX for the purpose unaware of the PostScript `pathtext` procedure for typesetting text along an arbitrary path, as published in Adobe's 1985 Blue Book. However ...

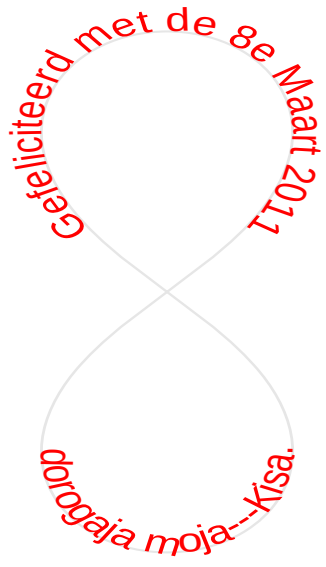
EPSF is an AnyTEXie's graphics companion

This note is not about OpenTypeFonts in PostScript in general, but only about how the congratulation was obtained via the use of `MinionPro-Regular`, an Adobe OpenTypeFont with Cyrillic glyphs, available in Acrobat Pro, which I use as PostScript interpreter.

Lemniscate

The equation for the lemniscate in polar coordinates (ϕ,r) reads $r^2 = 2a^2 \cos 2\phi$, a a constant.

Text along the lemniscate With the use of pathtext, to set text along an arbitrary path, as given by Program 11 p.167 of Adobe's Blue Book the following with congratulations in Dutch and sender in transcribed Russian, was easily obtained.




```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox:  -200  -300 200 310
%%BeginSetup
%%EndSetup
%%DocumentFonts: Helvetica
%%BeginProlog
(c:\\PSlib\\BlueBook.eps) run %pathtext
%%EndProlog
%
% Program ---the script---
%
/Helvetica 16 selectfont /a 90 def
%upper part of lemniscate
3 2.2 scale 90 rotate 0 0 moveto
45 -1 -45{/t exch def
/r sqrt2 a mul 2 t mul cos sqrt mul def
/x r t cos mul def
/y r t sin mul def
x y lineto
}for
gsave .9 setgray stroke grestore%paint path to the current page
gsave
1 0 0 setrgbcolor (Gefeliciteerd met de 8e Maart 2011) 41 pathtext
grestore
%lower part of lemniscate
newpath 0 0 moveto
45 -1 -45{/t exch def
/r 2 sqrt a mul 2 t mul cos sqrt mul def
/x r t cos mul neg def
/y r t sin mul def
x y lineto
}for
gsave .9 setgray stroke grestore%paint path to the current page
1 0 0 setrgbcolor (dorogaja moja---Kisa.) 91 pathtext
showpage
%%EOF
```

But ... how to do the above with Cyrillic glyphs?

Cyrillic by PostScript

I did not know on 8th March how to use Cyrillic in ASCII-biased PostScript. The above was my present, apart from two added photographs by Photoshop in the inner parts of the lemniscate.

But ... I did continue, I wanted to know how to use Cyrillic in PostScript. Is PostScript's CID multi-byte fonts machinery necessary?
For the use of Cyrillic in PostScript we have 2 problems:

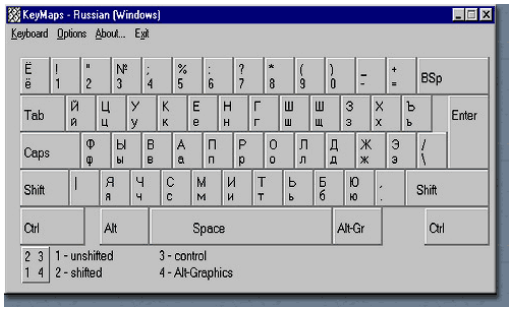
- what is the font name
- how to keyboard the characters.

The LRM 3, p.330 prompts the solution

> ... *It allows applications to specify how characters selected from a large character set are to be encoded. Some character sets*, especially OTF, *consist of more than 256 characters, including ligatures, accented characters, and other symbols required for highquality typography or non-Latin writing systems. Different encodings can select different subsets of the same character set.* ...

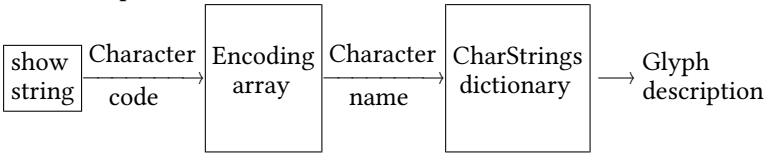
Font name What fonts are in PostScript available? `FontDirectory` does not provide the answer. I looked into the font directory of Acrobat Pro and found the `Minion-Pro-Regular` OTF. I read in the ATN `OpenType User Guide for Adobe Fonts' that when the name contains `Pro` the font has Cyrillic glyphs, which I verified with the MS accessory `Character Map' i.e. speciale tekens in Dutch.

Keyboarding Cyrillic The layout of a Russian keyboard is given below with at right the layout of my QWERTY keyboard.



With the help of `ReEncodeSmall` of the Blue Book Program 18, I reencoded the font `MinionPro-Regular` such that for example the ASCII code `8#141`, corresponding to the character a on the QWERTY keyword, is associated with the Russian `afii10086`, i.e. the glyph ф. (In ATN 5013 Adobe Standard Cyrillic Font Specification I found the Adobe names for the Cyrillic glyphs: capital Russian `A` is called `afii10017` etc.)

From the LRM 3, p.329



Encoding scheme for Type 1 fonts

> ... *For example, in the standard encoding vector used by Type 1 Latin-text fonts such as Helvetica, the element at index 38 is the name object ampersand. When* `show` *encounters the value 38 (the ASCII character code for &) as an element of a string it is printing, it fetches the encoding vector entry at index 38, obtaining the name object* `ampersand`. *It then uses* `ampersand` *as a key in the current font dictionary's CharStrings subdictionary and executes the associated charstring that renders the & glyph.* ...

The extra level in referencing in PostScript is more flexible than the (internal) 1-level of ASCII referencing in TEX. The lack of the encoding vector concept is a serious flaw in the otherwise parametric setup of TEX&METAFONT.

The encoding of the 33 Cyrillic letters and the most common punctuation marks is given in the procedure RUSvec in the program below.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Cyrillic (Changing the encoding vector; Blue Book Program~18 .p207)
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%BoundingBox: 0 530 700 620
%%CreationDate: March 2011
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\Bluebook.eps) run %used: ReEncodeSmall
/RUSvec [%Russian keybord a la MS word etc        %Cyrillic 33 letters
  8#101 /afii10038 8#141 /afii10086 %key A  a --> cyrillic letter f
  8#102 /afii10026 8#142 /afii10074 %... B  b --> cyrillic letter i
  8#103 /afii10035 8#143 /afii10083 %    C  c --> cyrillic letter es
  8#104 /afii10019 8#144 /afii10067 %    D  c --> cyrillic letter ve
  8#105 /afii10037 8#145 /afii10085 %    E  e --> cyrillic letter u
  8#106 /afii10017 8#146 /afii10065 %    F  c --> cyrillic letter a
  8#107 /afii10033 8#147 /afii10081 %    G  g --> cyrillic letter pe
  8#110 /afii10034 8#150 /afii10082 %    H  h --> cyrillic letter er
  8#111 /afii10042 8#151 /afii10090 %    I  i --> cyrillic letter sha
  8#112 /afii10032 8#152 /afii10080 %    J  j --> cyrillic letter o
  8#113 /afii10029 8#153 /afii10077 %    K  k --> cyrillic letter el
  8#114 /afii10021 8#154 /afii10069 %    L  l --> cyrillic letter de
  8#115 /afii10046 8#155 /afii10094 %    M  m --> cyrillic letter soft sign
  8#116 /afii10036 8#156 /afii10084 %    N  n --> cyrillic letter te
  8#117 /afii10043 8#157 /afii10091 %    O  0 --> cyrillic letter shcha
  8#120 /afii10025 8#160 /afii10073 %    P  p --> cyrillic letter ze
  8#121 /afii10027 8#161 /afii10075 %    Q  q --> cyrillic letter short i
  8#122 /afii10028 8#162 /afii10076 %    R  r --> cyrillic letter ka
  8#123 /afii10045 8#163 /afii10093 %    S  s --> cyrillic letter yeru
  8#124 /afii10022 8#164 /afii10070 %    T  t --> cyrillic letter ie
  8#125 /afii10020 8#165 /afii10068 %    U  u --> cyrillic letter ghe
  8#126 /afii10030 8#166 /afii10078 %    V  v --> cyrillic letter em
  8#127 /afii10040 8#167 /afii10088 %    W  w --> cyrillic letter tse
  8#130 /afii10041 8#170 /afii10089 %    X  x --> cyrillic letter che
  8#131 /afii10031 8#171 /afii10079 %    Y  y --> cyrillic letter en
  8#132 /afii10049 8#172 /afii10097 %    Z  z --> cyrillic letter ya
  8#173 /afii10039 8#133 /afii10087 %    {  [ --> cyrillic letter X
  8#175 /afii10044 8#135 /afii10092 %    }  ] --> cyrillic letter hard sign
  8#176 /afii10023 8#140 /afii10071 %    ~  ` --> cyrillic letter io
  8#42   /afii10047 8#47  /afii10095 %    "  ' --> cyrillic letter e
  8#74   /afii10018 8#54  /afii10066 %    <  , --> cyrillic letter be
  8#76   /afii10048 8#56  /afii10096 %    >  . --> cyrillic letter yu
  8#72   /afii10024 8#73  /afii10072 %    :  ; --> cyrillic letter zhe
  8#44   /semicolon 8#136 /colon     %    $  ^ --> semicolon   colon
  8#43   /afii61352 8#46  /question  %    #  & --> numero sign question mark
  8#77   /period     8#57  /comma     %    ?  / --> period      comma
] def
%
```

```
/width 19 def /tabstops{pop pop /k k 1 add def k width mul y moveto}def
%%EndProlog
/MinionPro-Regular /MPR RUSvec ReEncodeSmall
/k 0 def /y 600 def 0 y moveto
/Courier 16 selectfont
{tabstops}(ABCDEFGHIJKLMNOPQRSTUVWXYZ{}:><~"#?$^) kshow
/k 0 def /y 580 def 0 y moveto
/MPR 16 selectfont
{tabstops}(ABCDEFGHIJKLMNOPQRSTUVWXYZ{}:><~"#?$^) kshow
%
/k 0 def /y 555 def 0 y moveto
/Courier 16 selectfont
{tabstops}(abcdefghijklmnopqrstuvwxyz[];.,`'&/) kshow
/k 0 def /y 535 def 0 y moveto
/MPR 16 selectfont
{tabstops}(abcdefghijklmnopqrstuvwxyz[];.,`'&/) kshow
showpage
%%EOF
```

with result

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z { } : > < ~ " # ? $ ^
Ф И С В У А П Р Ш О Л Д Ь Т Щ З Й К Ы Е Г М Ц Ч Н Я Х Ъ Ж Ю Б Ё Э №. ; :
a b c d e f g h i j k l m n o p q r s t u v w x y z [ ] ; . , ` ' & /
ф и с в у а п р ш о л д ь т щ з й к ы е г м ц ч н я х ъ ж ю б ё э ? ,

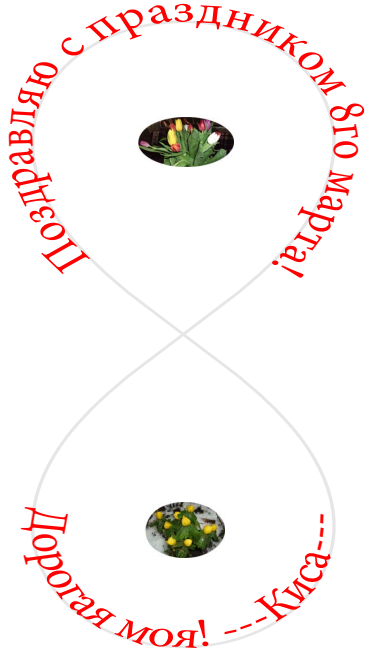
QWERTY keys (upper and lower case) with the Cyrillic glyphs underneath

A nice application of kshow this tabular printing of the glyphs; the more efficient xshow could have been used as well, though a little more verbose with the array of equal widths. Another option is to modify the width parameter of the font.

Result

The modified 8th March program with Cyrillic and the 2 photographs embedded in PostScript reads

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: 8 March surprise in Cyrillic
%%Author: Kees van der Laaan, kisa1@xs4all.nl
%%CreationDate: March 2011
%%BoundingBox:  -200  -300 200 310
%%DocumentFonts: MinionPro-Regular
%%BeginProlog
(c:\\PSlib\\Pslib.eps) run %Rusvec ReEncodeSmall
            % and pathtext (modified with centershow)
%%EndProlog
/MinionPro-Regular /MPR RUSvec ReEncodeSmall
/MPR 16 selectfont
%lemniscate upper part
/a 90 def
3 2.2 scale 90 rotate
0 0 moveto
45 -1 -45{/t exch def
/r sqrt2 a mul 2 t mul cos sqrt mul def
/x r t cos mul def
```



```
/y r t sin mul def
x y lineto
}for
gsave .9 setgray stroke grestore%paint the path, and restore the path
gsave
1 0 0 setrgbcolor (Gjplhfdkz. c ghfplybrjv 8uj vfhnf) 41 pathtext
grestore
%lemniscate lower part
newpath 0 0 moveto
45 -1 -45{/t exch def
/r sqrt2 a mul 2 t mul cos sqrt mul def
/x r t cos mul neg def
/y r t sin mul def
x y lineto
}for
gsave .9 setgray stroke grestore%paint the path, and restore the path
                                %for reuse by pathtext
1 0 0 setrgbcolor (Ljhjufz vjz ---Rbcf---) 85 pathtext
%embedding of converted .jpg photographs
gsave -40 150 translate (C:\\CyrillicinPS\\tulpen.eps) run
grestore
-35 -200 translate      (C:\\CyrillicinPS\\winteraconiet.eps) run
showpage
%%EOF
```

Note The positioning of embedded EPSF photographs I did by trial and error, quick and dirty. Insert the `.jpg` photographs in Photoshop, interactively, if you wish.

Conclusions

Using an OTF with Cyrillic glyphs was laborious and confusing, especially creating the encoding vector. More OTF's with Cyrillic would be nice.

The `RUSvec` encoding vector I expect to be applicable to any Adobe font with `pro` as part of the name, because the Russian keyboard and the `afii`... -names are general.

The inclusion of `.jpg` illustrations converted to EPSF by Adobe Photoshop is handy.

I don't know how the use the OTF MinionPro-Regular in TEX with the input via a Cyrillic keyboard, apart from the hazzle of the conversion of accompanying AFM into TFM, needed by TEX. The wired-in ASCII code table in TEX inhibits Cyrillic keyboarding.

TEX is becoming of age. In PostScript the AFM is not needed.

My wife was happy with the present.

Æsthetics and effectiveness of the message, cultural contexts?

- PostScript can be used gracefully, æsthetically, and effectively to set Cyrillic along an arbitrary path
- EPSF is an AnyTEXie's graphics companion.

TEXies should catch up

Literature

- Adobe's Red and Blue books. `http://www-cdf.fnal.gov/offline/PostScript/`
- AdobeTechnicalNote#5013, 18 February 1998. Adobe Standard Cyrillic Font Specification.

- OpenType User Guide for Adobe Fonts, October 2008. http://www.adobe.com/type/opentype
- AdobeTechnicalNote#5002, Encapsulated PostScript File Format Specification http://partners.adobe.com/public/developer/en/ps/5002.EPSF_Spec.pdf
- The Unicode Standard 6.0. 2010. http://www.unicode.org/versions/Unicode6.0.0/
  For use of Cyrillic in TEX my EuroBachoTEX 2002 paper
- Cyrillic and TEX — reappraisal of the WNCY font set.

(The keyboarding of Cyrillic is done by transcription which with WNCY fonts and the ligature mechanism yields the Cyrillic glyphs. I don't know how to emulate a Cyrillic keyboard in TEX. The assignment of charcode 98 to the letter b etc is internal to TEX, which I can't modify as mentioned in Ch8 The TEXBook: ... *Thus, b is 98 inside of TEX even when your computer normally deals with EBCDIC or some other non-ASCII scheme....* A little later in the chapter it is explained how to deal with a Norwegian keyboard with key æ, for example, via ligatures.
Even the Russian translation of the TEXbook, Все про TEX, does not explain how to keyboard Cyrillic, does not mention a Russified TEX, a serious omission IMHO.)

Note My BachoTEX 2010 programming pearl — typesetting Π-decimals along an spiral — does not make use of an explicit path.

Acknowledgements

Thank you Adobe for your maintained, adapted to LanguageLevel 3 since 1997, good old, industrial standard PostScript and Acrobat to view it, for your Photoshop which allows straightforward conversion to EPSF, Don Knuth for your stable plain TEX, Jonathan Kew for the TEXworks IDE, Hàn Thế Thành for pdfTEX.

Thank you Svetlana Morozova for the Russian keyboard layout and for prompting the Russian texts, and not in the least for being the person to be surprised by my present. Thank you MAPS editors for improving my use of English and Taco Hoekwater for procrusting my plain TEX note into MAPS format.

My case rests, have fun and all the best.

Kees van der Laan
Hunzeweg 57, 9893PB Garnwerd, Gr, NL
kisa1@xs4all.nl

Extending ConTEXt MKiV with PARI/GP

Abstract
This paper shows how to build a binding to PARI GP, the well known computer algebra system, for ConTEXt MKiV, showing also some examples on how to solve some common basic algebraic problems.

Keywords
LuaTeX, ConTeXt MARKIV, binding, PARI/GP

Introduction

PARI/GP[1] is a relatively small computer algebra system that comes as C library (`libpari`) and an interpreter (`gp`) for its own language (GP) built on upon the same library. Although it has discrete capabilities on symbolic manipulations, it has an extensive algebraic number theory module and hence it can do, due to the highly optimised C library, complex numeric calculations very quickly and accurately. In this paper we will show a way to 'extend' ConTEXt MKiV with PARI/GP and some examples on how to use this powerful library. PARI stands for 'Pascal ARIthmétique' (the very first choice was the Pascal language, dropped soon for C), while GP originally was GPC for 'Great Programmable Calculator', but the C was dropped for unknown reason. The current stable version is `2.3.4`.

Build the Lua binding

It's well known that ConTEXt MKiV uses LuaTEX as a typesetting engine, but maybe it's little known to the tex-user that Lua itself is used either as *embedded language* to enhance an application with a simple but powerful high level language (e.g. to build plug-ins) or as *glue language* to 'connect' several libraries, mostly written in C\C++ — exactly the same as in Sage[2], where the glue language is Python. In the latter case Lua is *extended* with the new libraries that become practically Lua modules (i.e. modules written in native Lua language) and they can be built in into the `lua` interpreter at compile time (as in the GSL Shell[3] program) or loaded at runtime, which is the case of the extension of this paper.

Most often it's necessary to write some C code that acts as an interface between the library and Lua: this process is called 'build the Lua binding for the library' and it's where the developer decides which symbols of the library (i.e. functions, classes, variables and constants) export to Lua and how they are seen from the Lua side (under which name, for example). This is a delicate phase, because one must know the conventions of the `lua` library on which the Lua language relies (the '`lua` Application Program Interface' or API), the API of the target library and write the appropriate C code for each symbol to export: for the C language these APIs are usually organized in so-called *header files* (with suffix `".h"`) that contain the declarations of each function, variable or constant that the library exposes — but not always all of them can be exported: the developer must consult the documentation to know which set of admissible symbols to export).

Luckily the `lua` API are completely listed in the Lua book[4] and they describe a simple and robust mechanism: basically every C function that wants to interact with the Lua interpreter uses a stack (a LIFO queue) to exchange data. The stack is modified by a relatively small set of functions that act on the *Lua state*, a global data structure that also keeps track of unused objects and calls the garbage collector when necessary. Hence every C function of the binding must only take care of calling the right function of the target library with the right arguments and to use the stack to exchange the in (input to the function) and/or out (output to the Lua interpreter) values. If the target library has many functions this is a long and tedious work, because most of the time the functions follow few common patterns and most of the binding code can be cut-and-pasted with few modifications, but on average the headers of the target library are difficult to read.

This is where SWIG enters the play. SWIG (*Simplified Wrapper and Interface Generator*, see[5]) is a program to help the developer to build bindings and, for some libraries, it can almost automatically build a binding by merely reading all headers files. SWIG reads a driver file, the so-called *interface file* `".i"`, and it executes its instructions. For `libpari` the instructions in the interface file `pari.i` are quite simple: basically 'read the headers and produce the binding'. This is for example the role of the `%include "pari/paritype.h";` instruction, that just says 'read the header paritype.h which is in the pari folder and write the binding code'; but we can also map some `libpari` functions into something else, as in

```
GEN uti_mael2(GEN m,long x1,long x2)
{return mael2(m,x1,x2);}
```

where the libpari macro mael2 is wrapped into the C function uti_meal2 for sake of simplicity.

The binding is then built with

```
swig  -lua pari.i
```

This is the complete interface file pari.i used under Linux 32 bit: the header files are in the sub-folder pari of the folder that contains the build script.

```
%module pari
%{
#include "pari.h"
ulong overflow;
%}

%ignore gp_variable(char *s);
%ignore setseriesprecision(long n);
%ignore  killfile(pariFILE *f);

%include "pari/paritype.h";
%include "pari/parisys.h";
%include "pari/parigen.h";
%include "pari/paricast.h";
%include "pari/paristio.h";
%include "pari/paricom.h";
%include "pari/parierr.h";
%include "pari/paridecl.h";
%include "pari/paritune.h";
%include "pari/pariinl.h";

%inline %{
GEN uti_mael2(GEN m,long x1,long x2)
{return mael2(m,x1,x2);}
GEN uti_mael3(GEN m,long x1,long x2,long x3)
{return mael3(m,x1,x2,x3);}
GEN uti_mael4(GEN m,long x1,long x2,long x3,
long x4)
{return mael4(m,x1,x2,x3,x4);}
GEN uti_mael5(GEN m,long x1,long x2,long x3,
long x4,long x5)
{return mael5(m,x1,x2,x3,x4,x5);}
GEN uti_mael(GEN m,long x1,long x2)
{return mael2(m,x1,x2);}
GEN uti_gmael1(GEN m,long x1)
{return gmael1(m,x1);}
GEN uti_gmael2(GEN m,long x1,long x2)
{return gmael2(m,x1,x2);}
GEN uti_gmael3(GEN m,long x1,long x2,long x3)
{return gmael3(m,x1,x2,x3);}
GEN uti_gmael4(GEN m,long x1,long x2,long x3,
long x4)
{return gmael4(m,x1,x2,x3,x4);}
GEN uti_gmael5(GEN m,long x1,long x2,long x3,
long x4,long x5)
{return gmael5(m,x1,x2,x3,x4,x5);}
GEN uti_gmael(GEN m,long x1,long x2)
{return gmael2(m,x1,x2);}
GEN uti_gel(GEN m,long x1)
{return gmael1(m,x1);}
GEN uti_gcoeff(GEN a,long i,long j)
{return gcoeff(a,i,j);}
GEN uti_coeff(GEN a,long i,long j)
{return coeff(a,i,j);}
%};
```

The binding is quite straightforward: almost every symbol of libpari has a counterpart in Lua with the same name; the symbols 'private' are exposed in paripriv.h which is not listed in pari.i — they aren't exported and hence they are not reachable from Lua.

The build script (for Linux) assumes the latest SWIG and PARI/GP installed under /opt/swig-2.0.2:

```
/opt/swig-2.0.2/bin/swig  -lua pari.i
gcc -ansi \
    -I./pari -I/opt/swig-2.0.2/include \
    -c pari_wrap.c  -o pari_wrap.o
gcc -Wall -ansi -shared -I./pari \
    -I/opt/swig-2.0.2/include -L./ \
    -L/opt/swig-2.0.2/lib pari_wrap.o \
    -lpari -lm -o pari.so
```

Once compiled, the pari.so is suitable to be loaded as Lua module with require("pari").

As a final note for this section, the same steps can be followed under Windows using MinGW[6] or with GUB[7] to cross-compile the library in a host system (Linux) for a target system (Windows) — as is the case of this paper, where the examples use a cross-compiled dll libpari.

Examples

Summations

As we said briefly in the introduction, PARI/GP has its own language GP, with more than 450 functions, and its interpreter, the gp program. Most of the time these functions are one-to-one with the functions exported by the library libpari, but sometimes there are some 'sugar syntactic' constructs for the sake of simplicity. In any case, libpari has the gp_read_str(char *) function that evaluates a GP sentence and returns the result, so that on the Lua side it's possible to use both the library and the GP language. The library is usually quicker than GP and it has a finer grain control — which usually also means that it's necessary to write more code.

In this first example, we will see how to calculate exactly a summation. The GP function is sum(X,a,b,expr,start) that stands for $\sum_{X=a}^{b} \mathrm{expr}(X,\cdot)$, where start is the initial value of $\mathrm{expr}(X,\cdot)$:

```
\startluacode
require("pari")
pari.pari_init(4000000,500000)
document = document or {}
document.lscarso= document.lscarso or {}
local function sum(X,a,b,expr,start)
   local avma = pari.avma
   local start = start or '0.'
   local res =
    pari.gp_read_str(string.format(
     "sum(%s=%s,%s,%s,%s)",X,a,b,expr,start))
   res = pari.GENtoTeXstr(res)
   pari.avma = avma
   return res
end
document.lscarso.sum = sum
\stopluacode

\starttext
\startTEXpage
\startformula
\sum_{k=0}^{30}\frac{4(-1)^k}{2k+1}=
  \ctxlua{context(document.lscarso.sum(
   "k",0,30,"4*(-1)^k/(2*k+1)","0"))}
\stopformula
\stopTEXpage
\stoptext
```

that gives

$$\sum_{k=0}^{30}\frac{4(-1)^k}{2k+1}=\frac{5863013579100197316985228 4}{1847292006410659792986502 5}$$

Let's explain the code step by step. First we need to load the module with `require("pari")` — assuming that the library is in the standard path or in the current folder (cfr. `CLUAINPUTS` in [8] for more details).

Next, we must avoid conflicts with other Lua functions. A common solution is to define a namespace (`document.lscarso` in this case), a local function (`sum(X,a,b,expr,start)`) and expose it with the namespace (`document.lscarso.sum= sum`). This is a general issue when one defines its own module, not only for PARI/GP — it's the same problem of redefining TEX macros.

There is another issue with PARI/GP. Like Lua, PARI/GP also uses a stack but it has not a garbage collector, and every time it makes a calculation the result is not deleted; after a while the stack is full and the process aborts. Luckily it's easy to clear the stack: at the beginning of every function it's sufficient to record the initial position on the stack with `local avma=pari.avma` and then reset the stack with `pari.avma=avma` just before the return statement of the function. This is an issue with `libpari`, because most of GP functions manage the stack correctly.

After these notes, calling the GP `sum` function is a matter of calling `gp_read_str(char *)` with the right formatted string which is trivial thanks to `string.format`, a standard LuaTEX function. Last but not least is `pari.GENtoTeXstr(GEN)`, a `libpari` function that translates a pari object (e.g a fraction) into a TEX expression. It's important to note that the result is exact because we have imposed with `start=0` that all the values are in $\mathbf{Q}$: if we want an approximated value just use `start=0.` and the result is

$$\sum_{k=0}^{30}\frac{4(-1)^k}{2k+1}=3.173842337190749408690224140$$

But we can do things a bit better. First, we want to control the *precision* of the result, i.e. how many digits to show. This is quite simple: the GP `default(.,.)` function can be used to get/set some internal constants and `realprecision` is what we need:

```
local function set_precision(prec)
   local avma = pari.avma
   local prec = math.floor(prec+0.5) or 28
   local res = pari.gp_read_str(
    string.format("default(realprecision,%s)",
      prec))
   res = pari.GENtostr(pari.gp_read_str(
    "default(realprecision)"))
   pari.avma = avma
   return res
end

local function get_precision(prec)
   local avma = pari.avma
   local res = pari.GENtostr(
    pari.gp_read_str(
     "default(realprecision)"))
   pari.avma = avma
   return res
end
```

Once we have the notion of precision, we can extend the summation to 'infinity', i.e. until the partial sums are stable within the precision. Of course this depends on the character of the series — in our case it's an alternating series. For this kind of series GP has the `sumalt(X=a,expr)` function that does the job:

```
local function sum_alternate(X,a,expr,prec)
   local avma = pari.avma
   local gp = document.lscarso.get_precision
   local oldprec = gp(prec)
   document.lscarso.set_precision(prec)
   local res=pari.GENtostr(pari.gp_read_str(
    string.format("sumalt(%s=%s,%s)",
                          X,a,expr)))
   document.lscarso.set_precision(oldprec)
   pari.avma = avma
```

```
   res=string.gsub(res,"(%d)","%1\\hskip0sp")
   return res
end
```

We can hence try to calculate the series with a precision of `800` digits:

```
\startformula
\sum_{k=0}^{\infty}\frac{4(-1)^k}{2k+1}=
\stopformula
\ctxlua{context(
  document.lscarso.sum_alternate(
  "k",0,"4*(-1)^k/(2*k+1)",800))}
```

Given that the result is quite long (see fig.1) with `string.gsub(res,"(%d)","%1\\hskip0sp")` we insert an invisible skip to help TEX to break the expression.

$$\sum_{k=0}^{\infty}\frac{4(-1)^k}{2k+1}\approx$$

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611738193261179310511854807446237996274956735188575272489122793818301194912983367336244065664308602139494639522473719070217986094370277053921717629317675238467481846766940513200056812714526356082778577134275778960917363717872146844090122495343014654958537105079227968925892354201995611212902196086403441815981362977477130996051870721134999999837297804995105973173281609631863674

Figure 1. Evaluation of an alternating series with 800 digit precision.

Of course this is a well known series: from $\arctan(1)=\frac{\pi}{4}$ one can calculate the Taylor expansion of $\arctan(x)$ around $x=0$ with `taylor(expr,x)`:

```
local function taylor(expr,x)
   local avma = pari.avma
   local res = pari.gp_read_str(
    string.format("taylor(%s,%s)",expr,x))
   res = pari.GENtoTeXstr(res)
   pari.avma = avma
   return res
end

$\mathrm{arctan}(x)=$
\startformula
\ctxlua{context(document.lscarso.taylor(
        "atan(x)","x"))}
\stopformula
```

i.e.

$$\arctan(x)=x-\frac{1}{3}x^3+\frac{1}{5}x^5-\frac{1}{7}x^7+\frac{1}{9}x^9-\frac{1}{11}x^{11}+\frac{1}{13}x^{13}-\frac{1}{15}x^{15}+O(x^{16})$$

The series is convergent in $x=1$ (there are several proofs about this, e.g. see [9]), hence

$$4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}=\sum_{k=0}^{\infty}\frac{4(-1)^k}{2k+1}=4\frac{\pi}{4}=\pi \quad .$$

It's important to note that theoretically this series has a slow convergence to π (it's hence a bad choice to calculate π) but *practically* it can be used with PARI/GP to give quickly an high precision result — this is the power of the library.

Before continuing, let's consider this summation:

```
\startformula
\sum_{k=0}^{3}\frac{1}{x^2+k}=
\ctxlua{context(document.lscarso.sum(
  "k",0,3,"1/(x^2+k)","0"))}
\stopformula
```

that gives

$$\sum_{k=0}^{3}\frac{1}{x^2+k}=\frac{4x^6+18x^4+22x^2+6}{x^8+6x^6+11x^4+6x^2}$$

PARI/GP is also capable of some symbolic calculations — it's not only a numeric library.

Continued fractions

A simple *finite* (canonical) continued fraction is a rational number q given by

$$q=a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{\ddots+\cfrac{1}{a_n}}}}$$

where a_0 is an integer and $a_{j,j>0}$ are strictly positive integers. Every rational number can be expressed with a finite continued fraction; if we consider a succession of finite continued fractions for $n\to\infty$ we have an *infinite* (canonical) continued fraction, and every irrational number has an unique infinite continued fraction. For a finite c.f. $[a_0,a_1,a_2,\ldots,a_n]$ the rational number given

by calculating all the intermediate fractions is usually termed as p_n/q_n i.e. $[a_0,a_1,a_2,...,a_n] = \frac{p_n}{q_n}$. For example $[0,3] = \frac{1}{3}$ and it's possible to show that p_n/q_n is the fraction in lowest terms. The GF `contfrac` function calculates (the vector of) the continued fraction of a rational number, while `contfracpnqn` given a (finite vector of) continued fraction returns p_n, q_n but the interesting point here is to use, given a *real* number with a fixed precision, the continued fraction to find its best rational approximation. The `libpari bestappr(x,A)` function calculates exactly what we need:

```
local function bestappr(x,A)
   local avma = pari.avma
   local x = tostring(x) or nil
   local A = math.floor(A+0.5)
   local res, bestx
   if x == nil then return nil,nil end

   bestx = pari.bestappr(pari.geval(
     pari.strtoGENstr(x)),
     pari.geval(
      pari.strtoGENstr(tostring(A))))
   res = {}
   res[1] = pari.GENtostr(bestx)
   res[2] = pari.GENtostr(
            pari.uti_gel(bestx,1))
   res[3] = pari.GENtostr(
            pari.uti_gel(bestx,2))
   pari.avma = avma
   return res[1],res[2],res[3]
end
```

Note that the return value is an array with 3 components, namely $p_n/q_n, p_n, q_n$. We also use `pari.uti_gel`, the *wrapped* version of `libpari gel` function, to access an array by components.

Instead of an arbitrary real number, we choose π because the libpari `mppi(long)` function gives π with the required precision .

```
\startluacode
local collect = {}
local avma = pari.avma
local prec = 800
document.lscarso.set_precision(prec)
avma = pari.avma
local pi = pari.mppi(prec)
local pi_str = pari.GENtostr(pi)
pari.avma = avma
--print("=====>pi:",pi_str)
for d = 4,50000,1 do
   res,num,den =
     document.lscarso.bestappr(pi_str,d)
   collect[res] = {num,den,d}
end
context("\\starttabulate[|l|l|]")
context("\\HL")
context(string.format(
         "\\NC %s \\NC %s \\NC\\NR",
        "fraction","approx. value"))
context("\\HL")
for k,v in pairs(collect) do
   print( k, v[1]/v[2],v[3])
   -- context(k, v[1]/v[2],v[3])
   context(string.format(
    "\\NC %s \\NC %s \\NC\\NR",k,v[1]/v[2]))
end
context("\\stoptabulate")
\stopluacode
```

Note that we use p_n,q_n as a key for the dictionary `collect`, so we have just the set of results – i.e. we drop the same best approximations for different denominators. For a precision of 800 digits and a range of denominators between 4 and 50000 we have hence:

fraction	approx. value
333/106	3.1415094339623
104348/33215	3.1415926539214
16/5	3.2
13/4	3.25
22/7	3.1428571428571
355/113	3.141592920354
19/6	3.1666666666667
103993/33102	3.1415926530119

where the approx. values are due to the Lua floating point math.

Equations

Solving numeric equations in PARI/GP required more attention than other packages. The `solve(X=a,b,expr)` GP function implements a very good algorithm but it works with one variable only and it fails if `expr` is not defined in [a,b] and it hasn't a *variation* in [a,b]. This Lua wrapper `solve` tries to ensure that at in [a,b] there is a variation evaluating the sign of `expr(a)*expr(b)`:

```
function solve(expr,X,a,b,prec)
   local av = pari.avma
   pari.gp_read_str(
    string.format(
     "default(realprecision,%s)",prec))
   local tr,res
   pari.gp_read_str(string.format("f(%s)=%s",
                    X,expr))
   tr = pari.gp_read_str(
    string.format(
     "if(f(%s)*f(%s)<0,1,0)",a,b))
   tr = pari.GENtostr(tr)
```

```
   tr = tonumber(tr)
   res = nil
   if (tr==1) then
    local expr=string.format(
     "solve(%s=%s,%s,%s)",X,a,b,expr)
    res = pari.gp_read_str(expr)
    res = pari.GENtostr(res)
   end
   return res,
    pari.GENtoTeXstr(
     pari.strtoGENstr(expr))
end
```

The next code tries to solve

$$x^5 + x^3 \arctan(x) + 2x^2 + x + 1 = 0$$

for $x \in [-100,100]$ with a precision of 12 digits:

```
\startluacode
local solve = document.lscarso.solve
for a=-100,99,1 do
 local res,TeX,aa,bb =
   solve('x^5+atan(x)*x^3+2*x^2+x+1',
    "x",a,(a+1),12)
 if res  ~= nil then
  context(string.format(
   "$%s\\approx 0$ \\crlf
     for $x\\approx%s$\\par",
    TeX,res))
 else
     -- print("TeX="..TeX)
 end
end
\stopluacode
```

We have hence:
$x^5 + atan(x) * x^3 + 2 * x^2 + x + 1 \approx 0$
for $x \approx -1.47704735548$

PARI/GP has a rich set of functions for polynomials, and `solve` is not necessarily the best choice to find the roots of multivariate polynomials; the next example will show how to draw the real roots of P[X,Y] with a given precision in a square region $[a,b] \times [a,b]$. First of all, we need to understand that with a fixed precision there is also an associated `zero`: with `precision=12` then `zero=1E-96`. Next, PARI/GP finds the complex roots of a univariate polynomial, so we need a `get_value` wrapper to evaluate $P(x,y)$ for $y \in [a,b]$ (with a given precision), so we have an expression in the x variable that we will consider as a polynomial P[X]:

```
local function get_value(expr,X,a,prec)
   local avma = pari.avma
   pari.gp_read_str(string.format(
    "default(realprecision,%s)",prec))
   pari.gp_read_str(string.format(
    "%s=%s",X,a))
   local res = pari.gp_read_str(
    string.format("eval(%s)",expr))
   res  = pari.GENtostr(res)
   pari.avma = avma
   return res
end
```

The `polroots` function evaluates the roots and returns an array of roots where each root is separated into the real and complex components:

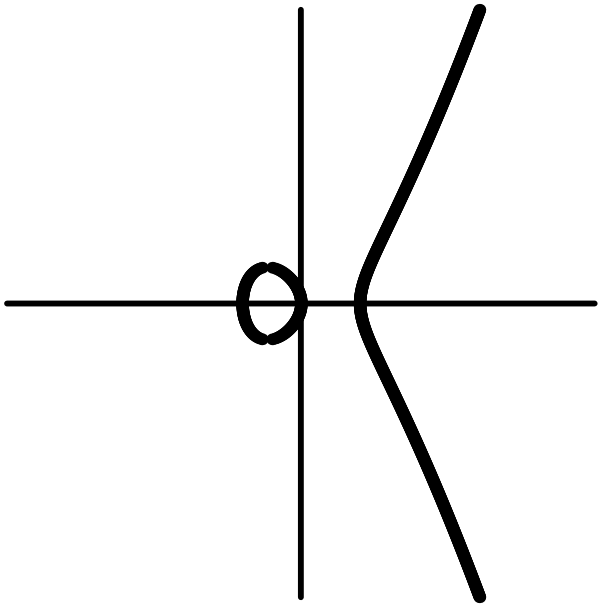
```
local function polroots(poly,prec)
   local avma = pari.avma
   pari.gp_read_str(string.format(
     "default(realprecision,%s)",prec))
   local poly = tostring(poly)
   local prec = tonumber(prec)
   local degree = pari.degree(
      pari.geval(pari.strtoGENstr(poly)))
   local roots = pari.roots(
    pari.geval(pari.strtoGENstr(poly)),prec)
   local res ={}
   for i=1,degree do
     local real_part,im_part =
      pari.GENtostr(pari.uti_gel(
      pari.uti_gel(roots,i),1)),
      pari.GENtostr(pari.uti_gel(
      pari.uti_gel(roots,i),2))
     res[#res+1]={real_part,im_part}
   end
   pari.avma = avma
   return res
end
```

Last we need to iterate y over $[a,b]$ and find the roots of P[X]. Instead of producing a table, we plot the value by a MetaPost page:

```
\startluacode
local poly = "x^3-x-y^2"
local step= 1/2^6
local results = {}
local limit = 5
local zero = '0.E-96'
local prec = 12
get_value = document.lscarso.get_value
polroots = document.lscarso.polroots
context("\\startMPpage")
context("pickup pencircle scaled 0.1pt;")
context(string.format("draw (-%s,0)--(%s,0);",
    limit,limit))
context(string.format("draw (0,-%s)--(0,%s);",
    limit,limit))
context("pickup pencircle scaled 0.2pt;")
for y=-limit,limit,step do
 local poly_x = get_value(poly,'y',y,prec)
 -- print("poly_x="..poly_x,y)
```

```
 local roots = polroots(poly_x,prec)
 for _,root in pairs(roots) do
  local real,imag = root[1],root[2]
  -- print("real=".real,"imag=".imag)
  if imag == zero then
     if real ==  zero then real = '0' end
     --print(string.format("(%s,%s)",real,y))
     context(string.format("draw (%s,%s);",
       real,y))
  end
 end
end
context("\\stopMPpage")
\stopluacode
```

With a precision of 12 digits and a square region of [−5,5] we have then :



Implicitization of a cubic bezier curve

The next and last example will show how to find, given $\mathbf{p},\mathbf{c_1},\mathbf{c_2},\mathbf{q}$ the points of a cubic Bezier curve in parametric form ($\mathbf{p}$ start point, $\mathbf{c1}$ and $\mathbf{c2}$ control points and $\mathbf{q}$ end point), a polynomial P[X,Y] that is the implicit form of the curve. Given the parametric form of a cubic Bezier $\mathcal{C} \in \mathbf{Q}$

$$\mathcal{C} = \Big\{ (1-t)^3\mathbf{p} + 3(1-t)^2 t\mathbf{c_1} + 3(1-t)t^2\mathbf{c_2} + t^3\mathbf{q}, \quad t \in [0,1] \Big\}$$

for a point $(x_t,y_t) \in \mathcal{C}$ we have

$$x_t = a_3t^3 + a_2t^2 + a_1t + a_0 = a(t)$$
$$y_t = b_3t^3 + b_2t^2 + b_1t + b_0 = b(t)$$

Following Sederberg([10], chap. "Algebraic Geometry for CAGD"), let

$$f = f(t,x) = a(t) - x$$
$$g = g(t,y) = b(t) - y$$

and

$$h_1(t,x,y) = (a_3g - b_3f)$$
$$h_2(t,x,y) = (a_3t + a_2)g - (b_3t + b_2)f$$
$$h_3(t,x,y) = (a_3t^2 + a_2t + a_1)g - (b_3t^2 + b_2t + b_1)f$$

In PARI/GP every indeterminate has an order and the first indeterminate is x, so it's better rename (t,x,y) → (x,X,Y) so that each h_j can be seen as a polynomial $(h_j[X,Y])[x]$ with at most degree 2 with respect to x. If we are able to find $h_1[x] = h_2[x] = h_3[x] = 0$ (the *null polynomial* of $\mathbf{Q}[x]$) then we have found the implicit form of our curve. It can be demonstrated that, if $h_{j,n}$ is the coefficient of x^n of h_j,

$$\begin{pmatrix} h_{1,2}[X,Y] & h_{1,1}[X,Y] & h_{1,0}[X,Y] \\ h_{2,2}[X,Y] & h_{2,1}[X,Y] & h_{2,0}[X,Y] \\ h_{3,2}[X,Y] & h_{3,1}[X,Y] & h_{3,0}[X,Y] \end{pmatrix} \begin{pmatrix} x^2 \\ x \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

if and only if

$$\begin{vmatrix} h_{1,2}[X,Y] & h_{1,1}[X,Y] & h_{1,0}[X,Y] \\ h_{2,2}[X,Y] & h_{2,1}[X,Y] & h_{2,0}[X,Y] \\ h_{3,2}[X,Y] & h_{3,1}[X,Y] & h_{3,0}[X,Y] \end{vmatrix} = 0$$

and hence this determinant is our P[X,Y].

The code is quite long, but not complicated:

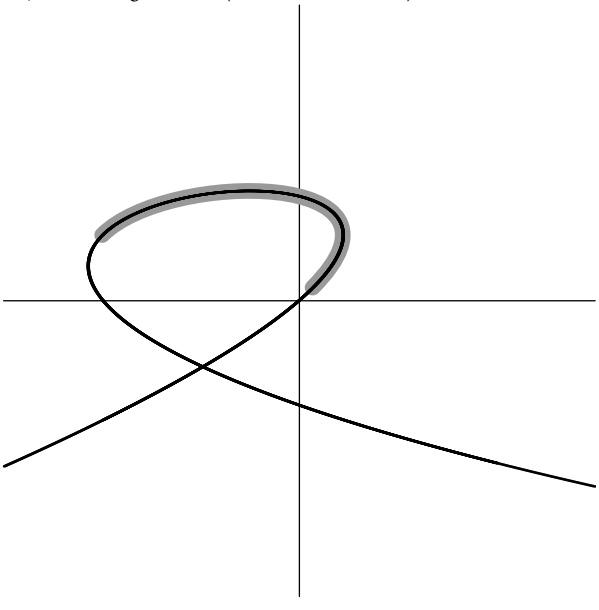
```
local function bezier_impl(p,c1,c2,q)
  local avma = pari.avma
  local f = string.format(
    "(1-t)^3*%s+3*(1-t)^2*t*%s+3*(1-t)*t^2*%s+t^3*%s",
    p[1], c1[1], c2[1], q[1])
  local g = string.format(
     "(1-t)^3*%s+3*(1-t)^2*t*%s+3*(1-t)*t^2*%s+t^3*%s",
     p[2], c1[2], c2[2], q[2])
  local fx =
   pari.gp_read_str(string.format("X-Pol(%s,x)", f))
  local gx =
   pari.gp_read_str(string.format("Y-Pol(%s,x)", g))
  fx =  pari.GENtostr(fx)
  gx =  pari.GENtostr(gx)
  local coeff_f_str =
```

```
 string.format("A=Vec(%s);B=if(poldegree(%s)==3,
  A,if(poldegree(%s)==2,[0,A[1],A[2],A[3]],
  if(poldegree(%s)==1,[0,0,A[1],A[2]],
  if(poldegree(%s)==0,[0,0,0,A[1]],[0,0,0,0])));B",
  fx,fx,fx,fx,fx)
local coeff_g_str =
 string.format("A=Vec(%s);B=if(poldegree(%s)==3,
 A,if(poldegree(%s)==2,[0,A[1],A[2],A[3]],
 if(poldegree(%s)==1,[0,0,A[1],A[2]],
 if(poldegree(%s)==0,[0,0,0,A[1]],[0,0,0,0])));B",
 gx,gx,gx,gx,gx)
local coeff_f = pari.gp_read_str(coeff_f_str)
local coeff_g = pari.gp_read_str(coeff_g_str)
local a3,a2,a1 =
 pari.uti_gel(coeff_f,1), pari.uti_gel(coeff_f,2),
 pari.uti_gel(coeff_f,3)
local b3,b2,b1 =
 pari.uti_gel(coeff_g,1), pari.uti_gel(coeff_g,2),
 pari.uti_gel(coeff_g,3)
local h1 =
  pari.gp_read_str(string.format("%s*(%s)-%s*(%s)",
   pari.GENtostr(a3), gx, pari.GENtostr(b3),fx))
local h2 =
  pari.gp_read_str(
   string.format("(%s*x+%s)*(%s)-(%s*x+%s)*(%s)",
   pari.GENtostr(a3), pari.GENtostr(a2), gx,
   pari.GENtostr(b3), pari.GENtostr(b2), fx))
local h3 =
  pari.gp_read_str(string.format(
   "(%s*x^2+%s*x+%s)*(%s)-(%s*x^2+%s*x+%s)*(%s)",
   pari.GENtostr(a3), pari.GENtostr(a2),
   pari.GENtostr(a1),gx, pari.GENtostr(b3),
   pari.GENtostr(b2), pari.GENtostr(b1),fx))
local h1_v = pari.gtovec(h1)
local h2_v = pari.gtovec(h2)
local h3_v = pari.gtovec(h3)
local idmat= pari.gp_read_str("idmat=matid(3)")
pari.gp_read_str(string.format("idmat[1,]=%s",
  pari.GENtostr(h1_v)))
pari.gp_read_str(string.format("idmat[2,]=%s",
  pari.GENtostr(h2_v)))
pari.gp_read_str(string.format("idmat[3,]=%s",
  pari.GENtostr(h3_v)))
idmat = pari.gp_read_str("idmat")
idmat_det = pari.gp_read_str("matdet(idmat)")
local PXY = pari.GENtostr(idmat_det)
local PxY =
 pari.gp_read_str(string.format("subst(%s,X,x)",PXY))
local Pxy =
 pari.gp_read_str(string.format("subst(%s,Y,y)",
  pari.GENtostr(PxY)))
local res = pari.GENtostr(Pxy)
local resTeX = pari.GENtoTeXstr(Pxy)
pari.avma = avma
return res,resTeX
end
```

For a curve $\mathcal{C}$ with $\mathbf{p} = (1,1)$, $\mathbf{c_1} = (10,10)$, $\mathbf{c_2} = (-10,10)$, $\mathbf{q} = (-15,5)$ we have

$$\begin{aligned} P[X,Y] = &-64X^3 + (2112Y + 312360)X^2 + \\ &(-23232Y^2 - 67920Y + 4711200)X + \\ &(85184Y^3 - 4440Y^2 - 5383200Y + 368000) \end{aligned}$$

It's easy to plot $\mathcal{C}$ with MetaPost (it's just `draw (1,1) .. controls(10,10) and (-10,10) .. (-15,5)`) so the next picture shows the MetaPost curve (thick, color gray) and the roots of P[X,Y] for $-15 <= x <= 15, -15 <= y <= 15$ (thin, color black).



Conclusion

One of the main beneficts of ConTEXt MKiV is the clear separation between Lua code and TEX code, and in this case it's a good thing that we can import a pari-lua script into ConTEXt MKiV without too much work to adapt it to the ConTEXt MKiV machinery — i.e. we have an high degree of code reuse. PARI/GP has also a nice TEX formatter, even if in some situations things are a bit raw. On the other side, solving numerical problems *always* requires some amount of theoretical analysis *before* doing the computation, as in the case of `solve` — in some circumstances PARI/GP abruptly aborts if it finds an error. Some computations can require a long time to finish, and given that ConTEXt MKiV is a multipass system a caching mechanism should be provided to solve these situations. Numeric results *can* (but they shouldn't) depend on the compiler and/or platform, but from this point of view it seems that PARI/GP is platform-independent.

Bibliography

[1] http://pari.math.u-bordeaux.fr.
[2] http://sagemath.org.
[3] http://www.nongnu.org/gsl-shell.
[4] http://www.inf.puc-rio.br/~roberto/pil2.
[5] http://swig.org.
[6] http://www.mingw.org.
[7] http://www.lilypond.org/gub.
[8] http://www.luatex.org/svn/trunk/manual/luatexref-t.pdf.
[9] http://en.wikipedia.org/wiki/Leibniz_formula_for_pi.
[10] tom.cs.byu.edu/~557/text/cagd.pdf

Luigi Scarso

Customised LaTeX page layout with LuaTeX

Abstract
The relationship between LaTeX's page layout parameters and the conventional desktop publishing (DTP) model of a page are explored and formulae to map between them are presented. A sample implementation of those formulae in Lua is provided, showing how to achieve customised page layouts in LuaTeX. The placement of crop marks is addressed, and a technique for preparing and adding them to typeset pages is discussed.

1 Introduction

Whilst LaTeX and its wealth of packages and facilities for typesetting are truly superb, I don't think it is unfair to say that it can be quite troublesome to achieve highly customised page layouts which need careful adjustment of the LaTeX parameters to control margins and page size. There are LaTeX packages, such as `geometry.sty`, which help you set the LaTeX layout parameters but sometimes it is nice to have access to the details to fine-tune them as you need to. In this paper, some equations which map LaTeX page layout parameters to the conventional desktop publishing (DTP) model of a page are presented and implemented through a simple Lua script for use with LuaTeX. In writing this paper I have to assume that you have a working installation of LuaTeX and that you know where to save and store the various file types discussed in this article: there is neither time nor space to address those issues here.

2 Problem definition: what are we aiming to do?

Suppose you want to produce a document which has a certain physical printed page width and page height, and you would like to achieve a layout such that your document's page(s) will be horizontally and vertically centred within the area defined by the PDF page size. Of course, the size of the PDF page is also something you want to control. Consider a typical business card which might be 85mm tall and 55mm wide. You want to create this card and have it centred within a PDF page which has, perhaps, 20mm of white space to the left and right of your card, and 10mm above and below, giving a PDF page width of $85 + (2 \times 20) = 125$mm and a height of $55 + (2 \times 10) = 75$mm. See Figure 1.

In addition, commercial printing companies may request that the pages in your PDF document are a certain size; e.g., for use with their imposition software used during preparation of "page impositions".

2.1 Setting the size of the PDF page

LuaTeX is derived from pdfTeX so you set the width and height of the PDF page using registers called `\pdfpagewidth` and `\pdfpageheight`. Returning to our business card example in Figure 1, to set the PDF page size to 125mm wide and 75mm tall we put the commands `\pdfpagewidth 125mm` and `\pdfpageheight 75mm` in our document, not forgetting that you may need to set `\pdfoutput=1`, depending on your TeX setup and distribution.

```
\documentclass[11pt,twoside]{article}
\pdfoutput=1
\pdfpagewidth 125mm
\pdfpageheight 75mm
\begin{document}
Your text here...
\end{document}
```

3 The "DTP design world" layout model

The usual way to think about layouts is, of course, as a series of enclosed boxes. Figure 2 generalises our business card example to show a typical setup for a "book" – but do remember that although I'm using the term "book", you should think of this as any document type. The outermost box is the size of the PDF page as defined by `\pdfpagewidth` and `\pdfpageheight`. Inside the PDF page box is the box defining the physical size of your "book" or document pages. Inside the box of your physical page is a non-printing area for margins which defines the boundary or enclosure for the live text area; i.e., the area in which text or other content will appear. And finally, within the live text area are the boxes that LaTeX uses to place the text on your page – such as the main text area, headers and footers.

4 LaTeX page parameters to achieve your layout

And so we reach the question: how do we define or calculate the LaTeX page parameters to achieve our desired "DTP design world" layout model? Figure 3 superim-

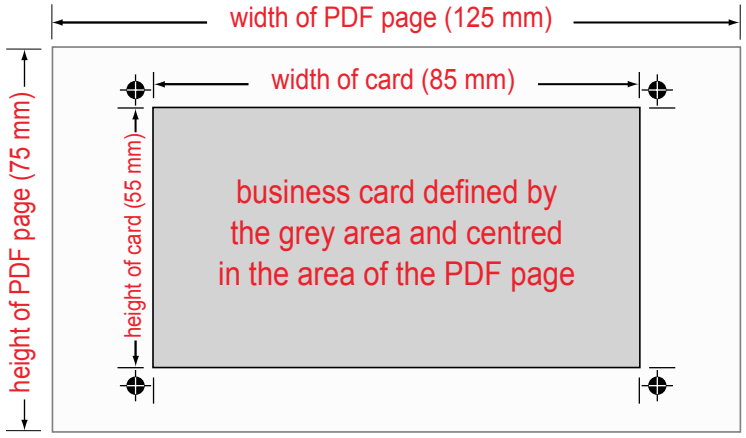


Figure 1. A business card centred within the area defined by the PDF document page. The crop marks define the boundaries of the card within the PDF document area.

poses the two world views: the LaTeX view and that of conventional desktop publishing, showing our page horizontally and vertically centred within a defined PDF document page. Separate graphics are shown for a left-hand page and a right-hand page, which, for left-to-right languages, usually implies books with even page numbers on the left-hand page and odd page numbers on the right-hand page.

Working from Figures 2 and 3 we can now write down some simple formulae to calculate LaTeX page parameters that will achieve our layout. Firstly, some definitions:

$$\begin{aligned} \mathrm{B_{PW}} &= \text{width of the book page} \\ \mathrm{B_{PH}} &= \text{height of the book page} \\ \mathrm{B_{OM}} &= \text{the Book Outer Margin} \\ \mathrm{B_{IM}} &= \text{the Book Inner Margin} \\ \mathrm{B_{TM}} &= \text{the Book Top Margin} \\ \mathrm{B_{BM}} &= \text{the Book Inner Margin} \end{aligned}$$

The following values control centring the book page within the PDF document page size:

$$\Delta\mathrm{X} = \frac{1}{2}(\mathrm{pdfpagewidth} - \mathrm{B_{PW}})$$
$$\Delta\mathrm{Y} = \frac{1}{2}(\mathrm{pdfpageheight} - \mathrm{B_{PH}})$$

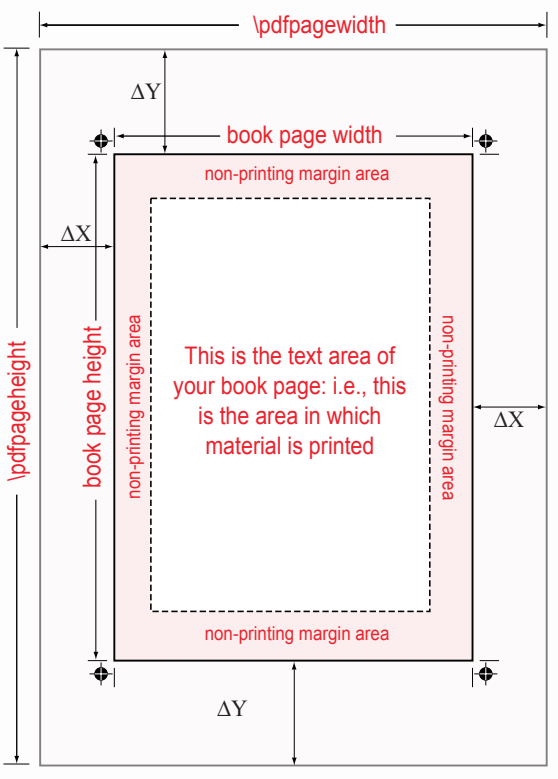


Figure 2. A generalised view of a "book" page: horizontally and vertically centred within the enclosing PDF document page area.

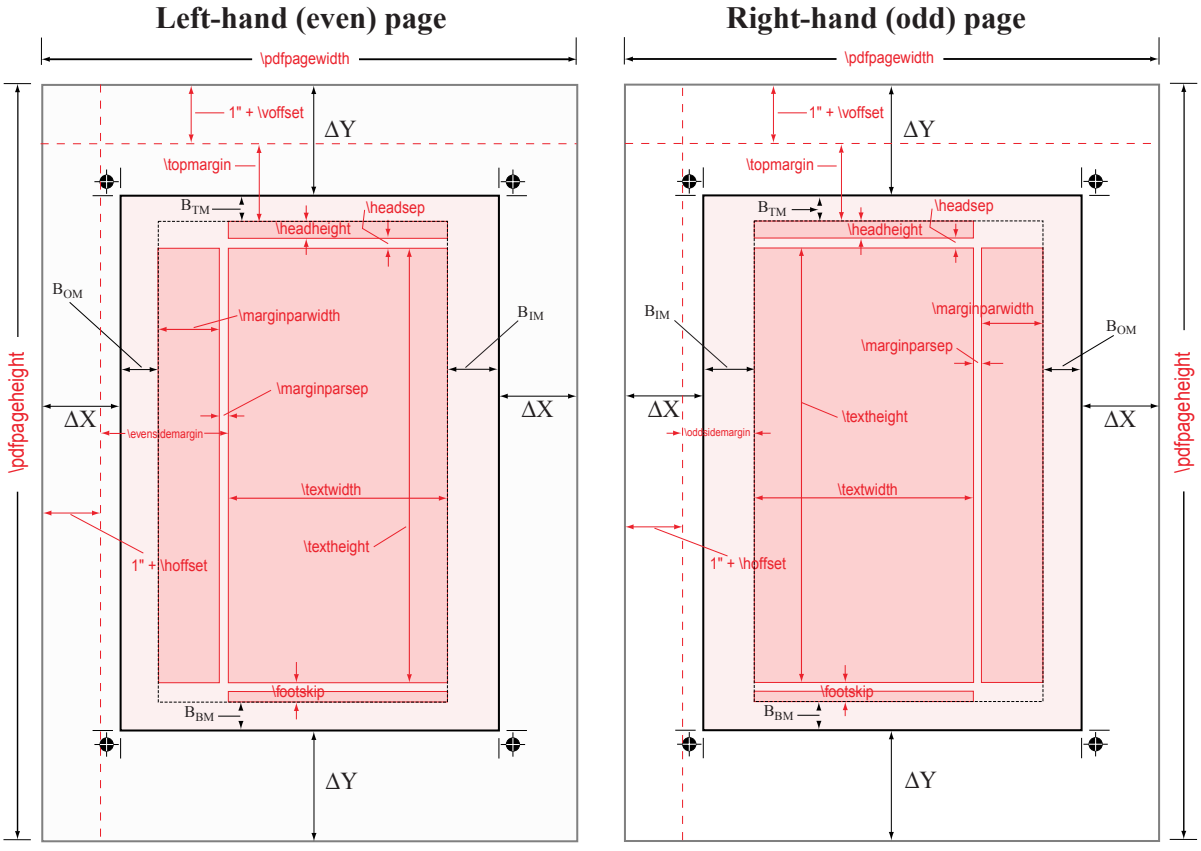


Figure 3. The LaTeX view of a page superimposed onto the conventional "desktop publishing" model to achieve a "book page" which is horizontally and vertically centred within the PDF document area. A separate graphic is shown for left- and right-hand pages.

4.1 Formulae for the width of the PDF document page

Starting with the left-hand (even-numbered) page shown in Figure 3:

$$\begin{aligned}\text{pdfpagewidth} = {} & \Delta\text{X} + \text{B}_{\text{OM}} \\ & + \text{marginparwidth} \\ & + \text{marginparsep} \\ & + \text{textwidth} \\ & + \text{B}_{\text{IM}} + \Delta\text{X}\end{aligned}$$

and

$$\begin{aligned}\text{pdfpagewidth} = {} & 1\text{inch} + \text{hoffset} \\ & + \text{evensidemargin} \\ & + \text{textwidth} \\ & + \text{B}_{\text{IM}} + \Delta\text{X}\end{aligned}$$

For the right-hand (odd-numbered) page shown in Figure 3:

$$\begin{aligned}\text{pdfpagewidth} = {} & \Delta\text{X} + \text{B}_{\text{IM}} \\ & + \text{textwidth} \\ & + \text{marginparsep} \\ & + \text{marginparwidth} \\ & + \text{B}_{\text{OM}} + \Delta\text{X}\end{aligned}$$

and

$$\begin{aligned}\text{pdfpagewidth} = {} & 1\text{inch} + \text{hoffset} \\ & + \text{oddsidemargin} \\ & + \text{textwidth} \\ & + \text{marginparsep} \\ & + \text{marginparwidth} \\ & + \text{B}_{\text{OM}} + \Delta\text{X}\end{aligned}$$

4.2 Formulae for the height of the PDF page

Clearly, the following holds true for left- and right-hand pages:

$$\begin{aligned}\text{pdfpageheight} = {} & 1\text{inch} + \text{voffset} \\ & + \text{topmargin} \\ & + \text{headheight} \\ & + \text{headsep} \\ & + \text{textheight} \\ & + \text{footskip} \\ & + \mathrm{B_{BM}} + \Delta\mathrm{Y}\end{aligned}$$

and

$$1\text{inch} + \text{voffset} + \text{topmargin} = \Delta\mathrm{Y} + \mathrm{B_{TM}}$$

5 Implementation using Lua and LuaTEX

The details above can be implemented in any TEX engine, whether they output PDF directly or you have to go through a DVI-to-PostScript driver. Of course, the direct use of `\pdfpagewidth` and `\pdfpageheight` would be ruled out for non-pdfTEX-based engines and for DVI–PostScript–PDF workflows you'd need to tell your DVI-to-PostScript driver how to set the paper size to the values we are specifying with `\pdfpagewidth` and `\pdfpageheight`. For example, the formulae above are easily programmed into, say, a Perl script or any other scripting language, to output a TEX file which sets the various LaTEX parameters. But for the purposes of this paper the implementation will be in Lua and LuaTEX.

5.1 Implementation overview

The goal is quite straightforward: the formulae above will be turned into a simple Lua script, say `pagecalcs.lua`, which will be run or called using LuaTEX's `\directlua{ }` primitive. `pagecalcs.lua` outputs a LaTEX file (`margins.tex`) which contains some code to define various LaTEX parameters such that your physical page area is centred within the PDF document page area.

5.2 Many choices

A quick inspection of the page formulae shows that the various parameters are all interrelated so the immediate question is which ones do you specify and which ones do you calculate? The answer really depends on your starting point; i.e., what type of document do you want to produce, which parameters do you want to define (assign values to) and which ones you want or need to calculate. The sample implementation, discussed below, is just *one* of the many possible variations. I have chosen the following setup which could be used for a typical business card or journal/book cover:

- set the following LaTEX values to zero: `\hoffset`, `\voffset`, `\marginparsep`, `\headheight`, `\marginparwidth`, `\headsep` and `\footskip`;
- calculate the following LaTEX parameters: `\textwidth`, `\topmargin`, `\oddsidemargin`, `\evensidemargin` and `\textheight`;
- input the "DTP design world" values for the paper (PDF page) size, the size of our physical "book" pages and the various white space margins surrounding the live text area (see Figure 2);
- generalise the margins to consider different values for the white space at the top, bottom, left and right of our live text area, which is quite typical for book designs. Of course, you can set them to be all the same, if you wish.

Clearly, these assumptions are hard-coded into `pagecalcs.lua` and a much more sophisticated implementation would support far more flexibility; but the purpose here is to demonstrate the basic ideas.

5.3 Using Lua

For our Lua script, `pagecalcs.lua`, we will use the following variable names which are rather long but have the benefit of being descriptive.

- `PaperWidth` = the value of `\pdfpagewidth`
- `PaperHeight` = the value of `\pdfpageheight`
- `BookPageWidth` = your document's page width
- `BookPageHeight` = your document's page height
- `BookOuterMargin` = $\mathrm{B_{OM}}$ as defined above
- `BookInnerMargin` = $\mathrm{B_{IM}}$ as defined above
- `BookTopMargin` = $\mathrm{B_{TM}}$ as defined above
- `BookBottomMargin` = $\mathrm{B_{BM}}$ as defined above

In the Lua code below, the variables names for the LaTEX parameters follow their LaTEX counterparts minus the leading '\'. For example,`\textwidth` will be referred to as `textwidth` and so forth. The following listing is `pagecalcs.lua`, just one of many calculation scenarios for one document type based on values we define and values we choose to calculate.

`pagecalcs.lua` defines a Lua function called `calcvals( )` which takes a single argument called `arg`. In Lua-speak `arg` will be a table so that when we call the function `calcvals({...})`, the data in braces `{...}` is passed in as the value of `arg` and will contain a number of key–value pairs. The values we will pass to `calcvals( )` are `PaperWidth`, `PaperHeight`, `BookPageWidth`, `BookPageHeight`, `BookOuterMargin`, `BookInnerMargin`, `BookTopMargin` and `BookBottomMargin`. Each of these values is accessed

pagecalcs.lua

```
function calcvals(arg)
    local OneInch=25.4
    local hoffset=0
    local voffset=0
    local marginparsep=0
    local headheight=0
    local marginparwidth=0
    local headsep=0
    local footskip=0
    local DeltaX = 0.5*(arg.PaperWidth - arg.BookPageWidth)
    local DeltaY = 0.5*(arg.PaperHeight - arg.BookPageHeight)
    local textwidth = arg.PaperWidth -(2*DeltaX) -
                          (arg.BookOuterMargin + arg.BookInnerMargin) -
                          (marginparsep + marginparwidth)
    local topmargin = DeltaY + arg.BookTopMargin -(OneInch + voffset)
    local oddsidemargin =  arg.PaperWidth - (OneInch + hoffset + textwidth +
                           marginparsep +  marginparwidth +
                           arg.BookOuterMargin+ DeltaX)
    local evensidemargin  =  arg.PaperWidth -(OneInch + hoffset + textwidth +
                          arg.BookInnerMargin + DeltaX)
    local textheight =  arg.PaperHeight -(OneInch + voffset + topmargin +
                           headheight + headsep + footskip +
                           arg.BookBottomMargin + DeltaY)
    local marg = assert(io.open("path_to_your_tex_setup/margins.tex","w"))
    marg:write("\\pdfpagewidth="..arg.PaperWidth.."mm\n")
    marg:write("\\pdfpageheight="..arg.PaperHeight.."mm\n")
    marg:write("\\newlength{\\deltax}\n")
    marg:write("\\setlength{\\deltax}{"..DeltaX.."mm}\n")
    marg:write("\\newlength{\\deltay}\n")
    marg:write("\\setlength{\\deltay}{"..DeltaY.."mm}\n")
    marg:write("\\newlength{\\bookpagetopmargin}\n")
    marg:write("\\setlength{\\bookpagetopmargin}{"..arg.BookTopMargin.."mm}\n")
    marg:write("\\newlength{\\bookpageheight}\n")
    marg:write("\\setlength{\\bookpageheight}{"..arg.BookPageHeight.."mm}\n")
    marg:write("\\newlength{\\bookpagebottommargin}\n")
    marg:write("\\setlength{\\bookpagebottommargin}{"..arg.BookBottomMargin.."mm}\n")
    marg:write("\\newlength{\\bookpagewidth}\n")
    marg:write("\\setlength{\\bookpagewidth}{"..arg.BookPageWidth.."mm}\n")
    marg:write("\\newlength{\\bookpageinnermargin}\n")
    marg:write("\\setlength{\\bookpageinnermargin}{"..arg.BookInnerMargin.."mm}\n")
    marg:write("\\newlength{\\bookpageoutermargin}\n")
    marg:write("\\setlength{\\bookpageoutermargin}{"..arg.BookOuterMargin.."mm}\n")
    marg:write("\\setlength{\\hoffset}{"..hoffset.."mm}\n")
    marg:write("\\setlength{\\marginparsep}{"..marginparsep.."mm}\n")
    marg:write("\\setlength{\\marginparwidth}{"..marginparwidth.."mm}\n")
    marg:write("\\setlength{\\textwidth}{"..textwidth.."mm}\n")
    marg:write("\\setlength{\\oddsidemargin}{"..oddsidemargin.."mm}\n")
    marg:write("\\setlength{\\evensidemargin}{"..evensidemargin.."mm}\n")
    -- Vertical parameters
    marg:write("\\setlength{\\voffset}{"..voffset.."mm}\n")
    marg:write("\\setlength{\\headheight}{"..headheight.."mm}\n")
    marg:write("\\setlength{\\headsep}{"..headsep.."mm}\n")
    marg:write("\\setlength{\\footskip}{"..footskip.."mm}\n")
    marg:write("\\setlength{\\textheight}{"..textheight.."mm}\n")
    marg:write("\\setlength{\\topmargin}{"..topmargin.."mm}\n")
    marg:flush()
    marg:close()
return DeltaX, DeltaY, textwidth,topmargin,oddsidemargin,evensidemargin,textheight
end -- function
```

using a standard Lua method for accessing table values, such as `arg.PaperWidth` and `arg.BookPageWidth`. The code also sets some LaTeX parameters to 0 (based on our input assumptions above) and defines `OneInch` as 25.4. Note that we are working with units in mm, just because it is convenient: hence 1 inch is 25.4 mm. Note the following:

- `local marg = assert(io.open("path_to...;`
- `calcvals( )` returns *multiple* values, a standard feature of Lua.

Of course, the output from `pagecalcs.lua`, `margins.tex`, will subsequently be input into our LaTeX document via `\input margins.tex`, hence you will need to output `margins.tex` into a location where your LuaTeX engine can find it.

6 The LaTeX side of things in LuaTeX

We have seen the Lua code to output `margins.tex` but, of course, we need some way to call the `calcvals( )` function defined in `pagecalcs.lua` and pass in the table `arg` containing our various values. We can do this from LuaTeX using one of the new primitives provided by this amazing new engine: `\directlua{...}`. One simple setup is using the standard LaTeX document class `article.cls`, as follows:

```
\documentclass[11pt,twoside]{article}
\input setpage
\setpage{300}{485}{250}{255}{10}{15}{20}{25}
\begin{document}
....
\end{document}
```

Of course, this should all be wrapped up into a proper LaTeX package but, again, my objective is to demonstrate the basic ideas. `\input setpage` inputs a file (`setpage.tex`) which defines a command `\setpage` that takes a number of arguments to define your custom PDF page, document page and margins and uses `\directlua{...}` to call our Lua function `calcvals( )`, which is itself stored in `pagecalcs.lua`. For example, `\setpage{300}{485}{250}{255}{10}{15}{20}{25}` would assign the following values (in mm):

- `PaperWidth` = 300mm
- `PaperHeight` = 485mm
- `BookPageWidth` = 250mm
- `BookPageHeight` = 255mm
- `BookOuterMargin` = B_{OM} = 10mm
- `BookInnerMargin` = B_{IM} =15mm
- `BookTopMargin` = B_{TM} = 20mm
- `BookBottomMargin` = B_{BM} =25mm

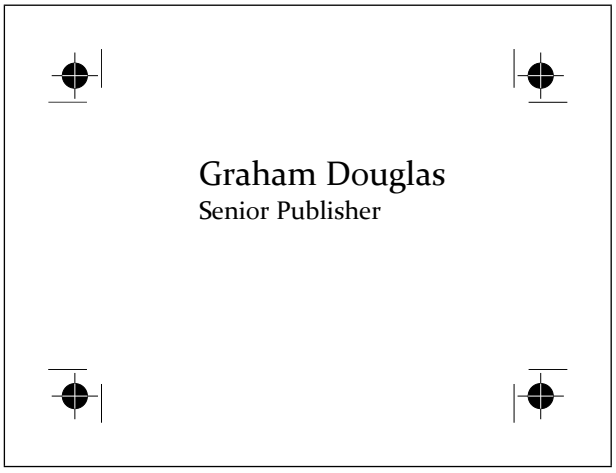


Figure 4. A business card

Here is the code for our file `setpage.tex` which again assumes that `pagecalcs.lua` is saved in a location where your LuaTeX engine can find Lua scripts.

setpage.tex

```
\def\setpage#1#2#3#4#5#6#7#8{%
\directlua {%
  pagecals, loaderror = loadfile("pagecalcs.lua")
  %good code would check the value of loaderror!
  pagecals()
  deltax, deltay, textwidth, topmargin,
  oddsidemargin,evensidemargin,textheight =
    calcvals({PaperWidth=#1, PaperHeight=#2,
              BookPageWidth=#3,
              BookPageHeight=#4, BookOuterMargin=#5,
              BookInnerMargin=#6, BookTopMargin=#7,
              BookBottomMargin=#8})
  pagevars={}
  pagevars["paperwidth"]=#1
  pagevars["paperheight"]=#2
  pagevars["bookpagewidth"]=#3
  pagevars["bookpageheight"]=#4
  pagevars["bookoutermargin"]=#5
  pagevars["bookinnermargin"]=#6
  pagevars["booktopmargin"]=#7
  pagevars["bookbottommargin"]=#8
  pagevars["deltay"]=deltay
  pagevars["deltax"]=deltax
  pagevars["textwidth"]=textwidth
  pagevars["topmargin"]=topmargin
  pagevars["oddsidemargin"]=oddsidemargin
  pagevars["evensidemargin"]=evensidemargin
  pagevars["textheight"]=textheight}
\input margins.tex\relax}
}
```

6.1 A business card

Just by way of an example, Figure 4 shows a business card created by `\setpage{125}{95}{85}{55}{0}{0}{0}{0}`. However, the code to generate the crop marks (see Section 7) at the corners is not shown here.

6.2 Anatomy of setpage.tex

The definition \def\setpage#1#2#3#4#5#6#7#8 is a fairly standard TeX affair to define a macro that takes 8 parameters, but the interesting bit starts with \directlua. Let me just say that, of course, the material in this paper could just as easily be implemented in Perl, or another scripting language, and the TeX engine could make system calls to run the script/interpreter of your choice to output margins.tex. Or, it could be implemented completely outside the TeX engine with the script being run manually. However, the use of LuaTeX's ability to run Lua code with \directlua makes for a very nice integration. For sure, this example does not even begin to indicate the world of possibilities that LuaTeX opens, for that you should visit luatex.org and grab a copy of the latest reference manual. But be warned, LuaTeX is highly addictive, utterly absorbing and quite damaging to other hobbies you may enjoy because it truly opens a whole new universe of typesetting solutions. Now, back to setpage.tex.

In the listing of setpage.tex the following lines are worth some explanation, especially if you are new to Lua or LuaTeX.

```
pagecals, loaderror = loadfile("pagecalcs.lua")
%good code would check the value of loaderror!
pagecals()
deltax, deltay, textwidth,topmargin,
oddsidemargin,evensidemargin,textheight =
   calcvals({PaperWidth=#1, PaperHeight=#2,
             BookPageWidth=#3,
             BookPageHeight=#4, BookOuterMargin=#5,
             BookInnerMargin=#6, BookTopMargin=#7,
             BookBottomMargin=#8})
```

The line pagecals, loaderror = loadfile ... uses Lua's method of "running" code stored in a file; it returns a function (here called pagecals) and an error value (here called loaderror), which you should check. Without going into detail, loadfile(...) "loads a Lua chunk from a file but does not run the chunk. Instead, it only compiles the chunk and returns the compiled chunk as a function." (Google, or see page 63 of *Programming in Lua* by Roberto Ierusalimschy, Second Edition, ISBN 85-903798-2-5).

In practical terms, loadfile(...) returns a function (here called pagecals) that you need to run in order to make calcvals(...) accessible. Don't worry about this Lua way of doing things, just accept it for present purposes. Once we have run pagecals() we can then call calcvals({...}) with the input values (as a Lua table).

The next few lines of setpage.tex create another Lua table, pagevars, in which we store the values returned by calcvals({...}) and the values of the \setpage parameters #1...#8. Finally, it inputs margins.tex into our document. The reason for creating pagevars will be outlined in Section 7.2.

6.3 A more complete example

By way of a more complete demonstration, the following example shows \setpage being used to define a custom page of 234mm tall × 156mm wide being output on a PDF page size of 180mm wide × 260mm tall (just enough to contain the crop marks, see Section 7). This example uses a number of additional packages including atbegshi.sty and fancyhdr.sty. In particular, fancyhdr.sty has been used to output the headers and footers to show that \setpage cooperates with fancyhdr.sty. The package atbegshi.sty is used to output the crop marks on the pages. See Figure 5. There is also some homegrown code to setup the crop marks (lines 9 to 12 of example.tex)

example.tex

```
\documentclass[11pt,twoside]{article}
\input setpages
\setpage{180}{260}{156}{234}{15}{25}{10}{10}
\usepackage{fontspec}
\usepackage{atbegshi}
\usepackage{fancyhdr}
\begin{document}
% start of crop mark code
\input pix
\startpix
\imbox{c:/crops.pdf}{crop}
\endpix
% end of crop mark code
\setmainfont[Ligatures=TeX,Numbers=OldStyle]
  {Constantia}
\fontsize{10}{11}\selectfont
\pagestyle{fancy}
\fancyhead{} % clear all header fields
\fancyhead[RO,LE]
  {\bfseries The performance of new graduates}
\fancyfoot{} % clear all footer fields
\fancyfoot[LE,RO]{\thepage}
\fancyfoot[LO,CE]{From: K. Grant}
\fancyfoot[CO,RE]{To: Dean A. Smith}
\renewcommand{\headrulewidth}{0.4pt}
\renewcommand{\footrulewidth}{0.4pt}

\textsc{but i must explain} to you how all this
  mistaken...

\end{document}
```

7 Crop marks

Many of the figures in this paper contain a small graphic at the corners of the area defined by the document pages. They are called *crop marks*, also referred to as "printers marks", "cut marks" or "trim marks", and are used to indicate the physical size of the final printed document pages. They are used during commercial printing activities, such as page imposition, colour separation, folding

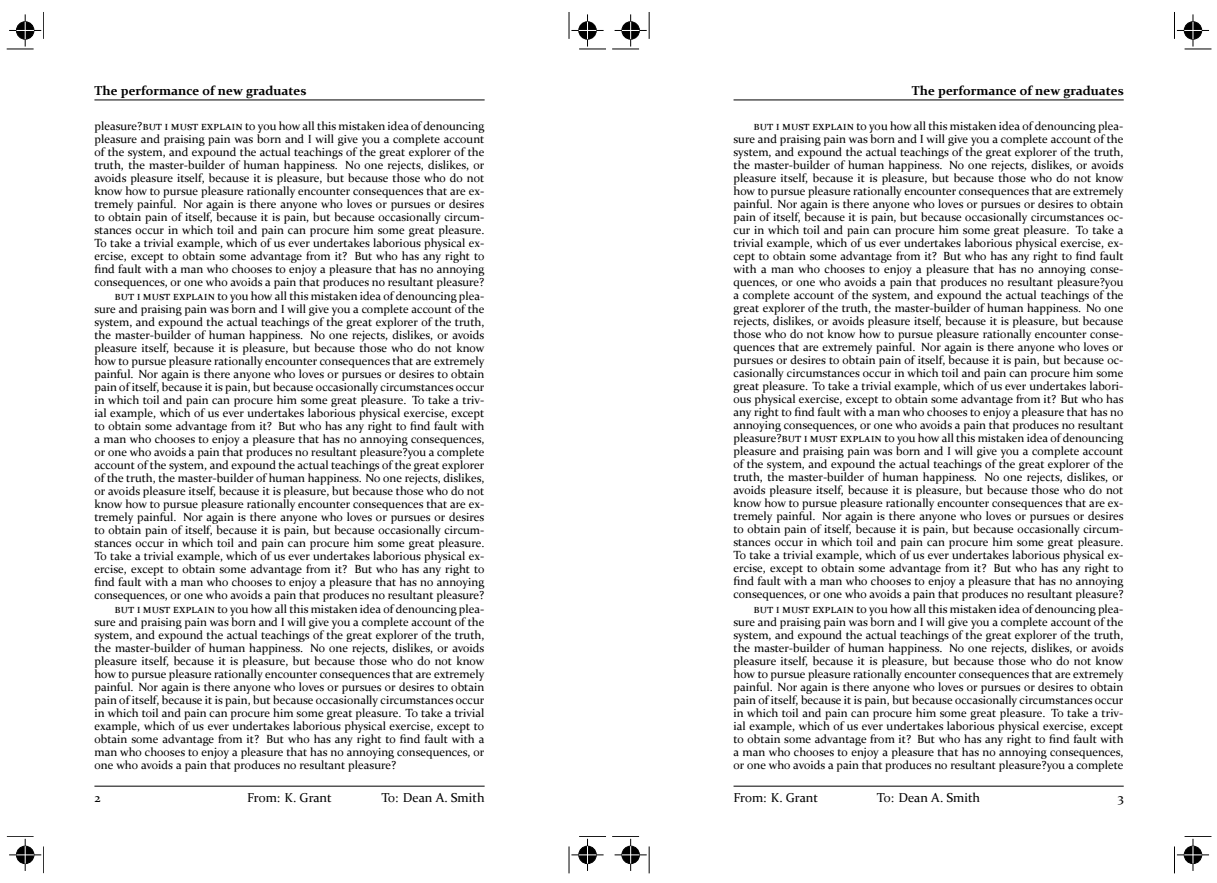


Figure 5. Using `setpage.tex` with `fancyhdr.sty` and `atbegshi.sty`.

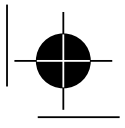


Figure 6. The crop mark produced by `crop.ps`.

and trimming. The physical appearance of crop marks will vary depending on the application used to generate the pages but, of course, with LuaTEX and other TEX engines you are free to create your own. With LuaTEX you can take advantage of the built-in METAPOST library, MPlib, to create crop marks with great precision. Figure 6 shows one design of crop mark, used by the author in various projects, which was created through some simple hand-rolled PostScript code (shown in listing `crops.ps` at the end of this article). To use this with LuaTEX or pdfTEX run the PostScript code through GhostScript or Adobe's Distiller to create a PDF file.

7.1 Crop marks and PDF XObjects

There are, of course, many possible techiques for getting crop marks placed on your pages, including PGF and TikZ, METAPOST, LaTEX packages and so forth. Here, I'll give an overview of the technique which has been used to generate crop marks on some of the figures in this paper. The crop mark shown in Figure 6 is stored in an external PDF file and embedded *just once* into the PDF generated by LuaTEX as a PDF object type called a *form XObject*, which is a "self-contained description of any sequence of graphics objects". The use of form XObjects helps to minimise the size of the PDF file: the data (PDF graphics operators) describing the XObject (our crop mark) are included into the PDF file just once. Instead of embedding the data multiple times, each page in your PDF file can "reference" the XObject by applying PDF operators to re-use it as part of the content of your pages. For example, by performing various coordinate transformations, such as translation or rotation, and then drawing the XObject into your transformed coordinate system.

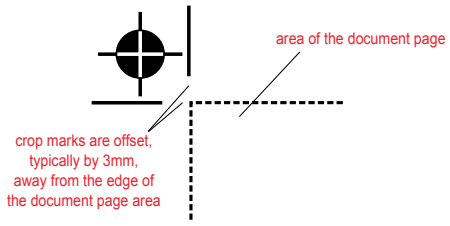


Figure 7. The location of crop marks relative to the document page.

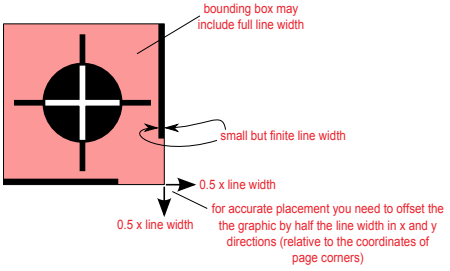


Figure 8. When placing an external graphic as a crop mark you may need to make micro-adjustments to account for the widths of lines.

7.2 Placing crop marks

Although crop marks are placed at the corners of the printed page, there are a couple of points to consider, especially when using the technique of an external graphic.

- crop marks are offset from the actual page area, typically by 3mm, so that they do not risk "contaminating" the actual printed area (see Figure 7);
- a more subtle point when placing crop marks contained in an external file is to take into account the actual width of the lines of the crop marks themselves and shift the positioning of the crop mark, relative to page corners, by half the width of the lines (see Figure 8);

Advanced readers will also observe that if you are involved in colour printing work, which requires colour separations, then of course, you may need to ensure that your crop marks appear on each colour separation/plate. However, this is beyond the scope of this paper and *definitely* something you should be discussing with the prepress department of your printing company. Additionally, the PDF specification makes specific provision for something called *Printer's mark annotations* which "...provide a mechanism for incorporating printer's marks into the PDF representation of a page, while keeping them separate from the actual page content." Again, this is beyond the scope of this paper and the interested reader is referred to the PDF specification (1.4 and later).

7.3 Getting crop marks onto the PDF page

Firstly, let me note that at the time of writing this article (April 2011), not only does LuaTeX support pdfTeX's facilities for embedding external PDF files but, in addition, it offers the built-in `epdf` library, thanks to the ongoing development work of Hartmut Henkel. The `epdf` library looks to provide a wonderful API for working with PDFs so by the time you read this article you'll likely have even more options at your disposal.

To place the crop marks you will, of course, need the (x, y) coordinates of the corners of your page, relative to some origin which is *usually* the bottom left-hand corner of the PDF document area. Although this origin is common in PDF documents the reader should be aware that it is not *required* by the PDF standard. Refer to the PDF specification for details on "user space", coordinate systems and the CropBox for detailed guidance.

In outline you need to:

1. embed your crop mark graphic as a form XObject;
2. determine the (x, y) coordinates of the corners of your page;
3. for each corner, apply some PDF operators to place and rotate the form XObject;
4. get the crop marks shipped out on every page.

Embedding a form XObject. This is quite straightforward using pdfTeX's primitive `\pdfximage`.

Calculating page-corner coordinates. Section 6.2 discussed `setpage.tex` and the creation of a Lua table called `pagevars` and here is where we can make use of it. Looking at `setpage.tex` you will see that `pagevars` stores the following values (in mm):

```
pagevars["paperwidth"]=#1
pagevars["paperheight"]=#2
pagevars["bookpagewidth"]=#3
pagevars["bookpageheight"]=#4
pagevars["bookoutermargin"]=#5
pagevars["bookinnermargin"]=#6
pagevars["booktopmargin"]=#7
pagevars["bookbottommargin"]=#8
pagevars["deltay"]=deltay
pagevars["deltax"]=deltax
```

We can re-use the values stored in `pagevars`, via `\directlua{ }`, to calculate our page corners, relative to the lower-left corner of the main PDF document. Remembering that `pagevars` stores our values in mm, the following Lua fragment calculates the values we need using the default PDF user space units of 72 points = 1 inch.

```
\directlua{
   dx = 72.0*(pagevars["deltax"]/25.4)
```

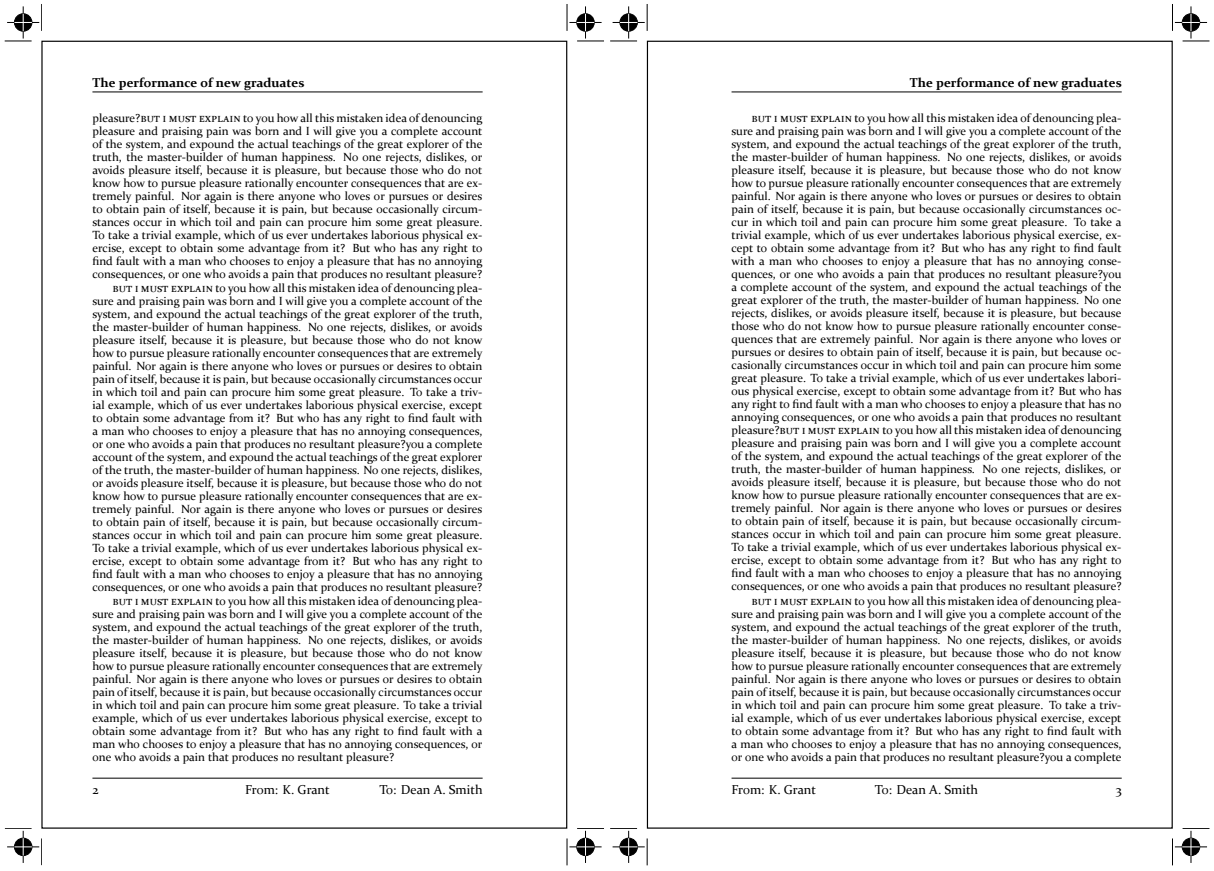


Figure 9. Using the `AtBeginShipout` command from `atbegshi` to draw a rectangle around the page area.

```
    dy = 72.0*(pagevars["deltay"]/25.4)
    bph=72.0*(pagevars["bookpageheight"]/25.4)
    bpw=72.0*(pagevars["bookpagewidth"]/25.4)
}
```

Working from Figures 2 and 3 and starting at the lower-left corner working clockwise, we can see that the page corner coordinates are $(\texttt{dx}, \texttt{dy})$, $(\texttt{dx}, \texttt{dy} + \texttt{bph})$, $(\texttt{dx} + \texttt{bpw}, \texttt{dy} + \texttt{bph})$, $(\texttt{dx} + \texttt{bpw}, \texttt{dy})$

Placing the form XObject. In addition to the commands inherited from pdfTEX, LuaTEX provides an extensive and rapidly developing set of APIs for working with PDF files, so you have a lot of choice in tackling this aspect of the problem. Here, we'll default to summarizing pdfTEX's `\pdfliteral page {<PDF code>}` which injects `PDF code` into the main PDF, establishing the origin as the lower-left corner of the PDF document page. This suits our needs because we already have our coordinates, from Section 7.3, relative to the lower-left corner of the main PDF document.

The graphic representing our crop mark, stored as a form XObject, clearly needs to be rotated as we move around the four corners of our page. Translation and rotation to establish a transformed coordinate system is a very standard operation within graphics and PDF work, typically achieved via matrix multiplications and won't be covered here. The PDF specification contains a very nice description of the relevant matrix maths and transformation of its coordinate systems and spaces.

PDF coordinate transformations. In PDF terms, for each page corner we need to:

1. modify the current transformation matrix using the PDF `cm` operator;
2. make a "call" to "paint" the embedded XObject using the PDF `Do` operator.

making sure, of course, that any modification to the current transformation matrix is local and enclosed within a PDF `q`...`Q` pair to save and restore the current PDF graphics state. To "execute" the `\pdfliteral ...` command which injects the PDF code to place and draw our crop mark, we use LuaTEX's `tex.print()` API call within a `\directlua { }` command. In the author's working code, this is achieved by calls such as this

```
tex.print("\\pdfliteral page{q "..s..
  " cm  /Im\csname pdf:crop\endcsname \space Do Q}")
```

By way of a small example, the following Lua fragment draws a rectangle around the area occupied by our pages within the actual PDF document area (see Figure 9 and compare to Figure 5).

```
% switch off current meaning of \\
\let\temp\\
\let\\\relax
\directlua{
  dx = 72.0*(pagevars["deltax"]/25.4)
  dy = 72.0*(pagevars["deltay"]/25.4)
  bph=72.0*(pagevars["bookpageheight"]/25.4)
  bpw=72.0*(pagevars["bookpagewidth"]/25.4)
  tex.print("\\pdfliteral page{q "..dx.."
    "..dy.." "..bpw.." "..bph.." re  S Q}")
}
```

7.4 Shipping crop marks on every page

And finally, you will need to ensure that your crop marks are shipped out on every page. One way to achieve this is to use the excellent `atbegshi` package by Heiko Oberdiek. In outline, you use `\AtBeginShipout{...}` with a `directlua{ }` command containing the Lua code required to place your crop marks. Here is a short fragment drawing a rectangle around our page. Note the `\let\temp\\` and `\let\\\relax` which temporarily disable LaTeX's definition of `\\` so that we can use this construct within `tex.print()`. See Figure 9.

```
\AtBeginShipout{
  \let\temp\\%
  \let\\\relax
  \directlua{
  dx = 72.0*(pagevars["deltax"]/25.4)
  dy = 72.0*(pagevars["deltay"]/25.4)
  bph=72.0*(pagevars["bookpageheight"]/25.4)
  bpw=72.0*(pagevars["bookpagewidth"]/25.4)
  tex.print("\\pdfliteral page{q "..dx.."
    "..dy.." "..bpw.." "..bph.." re S  Q}")
  }}
```

8 Conclusions and summary

The discussions and use of LuaTeX in this paper do not even begin to hint at the amazing versatility and potential of this incredible next-generation TeX engine. For that you should download and browse the latest LuaTeX Reference Manual from `luatex.org` and take a look at the available APIs. However, it is the author's hope that this paper has offered an interesting and practical starting point for anyone who wishes to explore LuaTeX.

8.1 Why I absolutely love LuaTeX

Here, I am grateful to the MAPS editors for allowing me a section of space for some personal reflection and indulgence. I first encountered LuaTeX when researching tools for typesetting my Arabic study notes and came across videos of conference presentations, by Hans Hagen and Idris Hamid, on River Valley's absolutely fabulous river-valley.tv website. I was transfixed and stunned by the truly beautiful and unbelievably sophisticated Arabic typography being presented by Hans and Idris. What was this amazing application? I recall being up to the early hours of the morning, on a work day..., Googling until I understood that the underlying TeX engine powering this incredible work was LuaTeX via, of course, the highly sophisticated ConTeXt package.

I had encountered the Lua language some years before and experimented with it for a number of projects, so I was aware of Lua's power, flexibility and ease of use as an embeddable scripting language with a wonderfully clean and straightforward C API, and certainly far easier to use than the APIs of more "heavyweight" scripting languages. Personally, I think the choice of Lua as the scripting language is perfect. For sure, we've all seen the mailing list or newsgroup arguments and threads spiralling into flame wars debating the merits of scripting language X or scripting language Y, with domain experts proposing complex arguments for and against a particular language. But LuaTeX is *here*, it is being *actively developed*, it *works* and it opens a truly *staggering* world of new solutions and opportunities to use the sophisticated algorithms of TeX exposed and accessible through a wonderfully simple but powerful scripting language.

The combination of a TeX-based typesetting engine coupled with OpenType fonts, UTF-8 input, exposure of TeX's internals, plug-ins through DLLs (on Windows) and all enabled through Lua as a scripting language was something I just had to explore. Well, nearly 18 months after my "discovery" of LuaTeX I have still not resumed my Arabic studies, continuing to follow the very latest updates of LuaTeX, compiling it from the latest source code as soon as any updates are available: sometimes daily. I am hooked, pure and simple! Be warned, LuaTeX is highly addictive, utterly absorbing and quite damaging to other hobbies you may enjoy. Happy TeXing.

8.2 Publication note

The material presented in this paper is based on independent work undertaken by the author in his spare time and does not necessarily reflect, represent or express any views or focus of interest or opinions of his employer the Institute of Physics or any of its group companies, including IOP Publishing Limited.

Graham Douglas, Senior Publisher
IOP Publishing, Dirac House, Temple Back,
Bristol BS1 6BE, UK
graham.douglas@readytext.co.uk

crops.ps

```
/ss   {setgray} def
/ssgs {/col exch def gsave col setgray} def
/gres {grestore} def
/MM {25.4 div 72 mul} def
%============================
% Define lengths
%============================
%offset from page
/TrimOffset 3 MM def
%length of trim mark
/TrimLength 8 MM def
/TotalLength TrimOffset TrimLength add def
%set the PostScript page size
<</PageSize
[TotalLength 0.25 add TotalLength 0.25 add]
>> setpagedevice
%Define Circle's parameters
%================================
%Centre of the circle
/CircleCentre TotalLength 2 div def
/Sqrt2 2 sqrt def
% Radius of circle
/CircleRadius  TotalLength Sqrt2 3 mul div def
% Define hair lengths
%============================================
/LengthOfBlackHair Sqrt2 1 sub Sqrt2 div neg
TotalLength mul TrimLength add 2 mul def
/LengthOfWhiteHair CircleRadius 2 mul def
%Positioning of black hairs
%============================================
/BlackHairGap TotalLength LengthOfBlackHair sub 2 div def
/WhiteHairGap TotalLength LengthOfWhiteHair sub 2 div def
%============================================
%Drawing procedures
%============================================
/DrawTrimLines        {
 0 ssgs
 0 TrimOffset moveto
 0 TrimLength rlineto
 TrimOffset neg 0 moveto
 TrimLength neg 0 rlineto
 stroke
 gres
} def

/DrawBlackHairs {
 0 ssgs
 CircleCentre neg BlackHairGap moveto
 0 LengthOfBlackHair rlineto stroke
 BlackHairGap neg CircleCentre moveto
 LengthOfBlackHair neg 0 rlineto stroke
 gres
} def

/DrawCircle {
 newpath
 0 ssgs
 CircleCentre neg CircleCentre CircleRadius
 0 360 arc
 gsave fill grestore stroke
 gres
} def

/DrawWhiteHairs {
 1 ssgs
 CircleCentre neg WhiteHairGap moveto
 0 LengthOfWhiteHair rlineto stroke
 WhiteHairGap neg CircleCentre moveto
 LengthOfWhiteHair neg 0 rlineto stroke
 gres
} def

/DrawAllMarks {
 0.5 setlinewidth
 DrawTrimLines
 DrawBlackHairs
 DrawCircle
 DrawWhiteHairs
} def

TotalLength 0.25 add 0 translate
-1 1 scale

TotalLength 0.25 translate
DrawAllMarks
showpage
```

LuaTEX Lua modules on Linux

Abstract
How to use the dynamic Lua module loading abilities in LuaTEX under Linux or similar systems.

Introduction

First, I should warn you that dynamic loading of modules is not quite trivial. It is simple enough if everything works as expected, but lots of problems can come up, and if you are not a programmer, it can be extremely confusing.

It is hoped that eventually some knowledgeable people will create ready-to-use packages that are then made available via the normal TEX distributions like TEX Live, MikTEX, or the ConTEXt minimals. But for now, you have to do it on your own (or find a programming friend to do it for you).

As in Thomas Schmitz' article elsewhere in this Maps, I will use the luasqlite3 module as an example. The latest version of this module is here: http://lua.sqlite.org/index.cgi/index

Building a module

On any Unix-like platform, you will first have to compile the module. Here is a step by step guide for Linux (I don't know how to compile modules for Windows or MacOSX, sorry), which is what I use, but for most Unix-like system the procedure is very close if not identical:

- Make sure you have both the `lua51` and `sqlite3` development packages installed. You need the development version of the packages because the module that we will build will be linked against those libraries. You can use your normal package manager for that, but the names of the packages may vary a little: my exact names on this Linux distribution where `liblua-devel` (version 5.1.4) and `libsqlite3-devel` (version 3.7.3).
- Then, download the module source from the URL given above, and unpack the zip file somewhere where you have write access. This will give you the directory `lsqlite3_svn08`.
- Move into the `lsqlite3_svn08` directory, and type `make`. With some luck, this is good enough, and you will now have `lsqlite3.so`. If `make` fails, it is likely because you have to adjust some of the variables in the top of the `Makefile`. Open `Makefile` in an editor and have a look. Usually, there are a few helpful comments included explaining you what you have to modify. Of course, if you cannot figure out what to do from that, you can always ask on the luatex mailing list (`luatex@tug.org`).

  Also, usually there is a file named `README` or `INSTALL` that can contain valuable hints.
- To verify that the created `lsqlite3.so` file is actually OK, type `make test`. The output should end with:

```
#### Test Suite finished.

479 Assertions checked. All Tests passed!
```

All the hard parts have now been done, we just have to copy the file to a more useful location.

Installation

for texlua

If LuaTEX runs as `texlua` (a.k.a. Lua interpreter mode), it will search for `lsqlite3.so` in exactly the same way as the standalone Lua does.

So, for that case, a simple `make install` will do the trick (you will probably need sudo rights). This will install the module in the correct system-wide spot and both `lua` and `texlua` will be able to find it.

Testing the standalone `texlua` is a simple case of, after `make install`, executing `make clean` to remove the generated files in the local directory and then using `texlua` to run the test suite instead of `lua`:

```
[... lsqlite3_svn08]$ texlua tests-sqlite3.lua
```

for typesetting

Having successfully created the `.so` file, now we also have to put it where LuaTEX can find it while typesetting.

To this end, LuaTEX uses a new kpathsea file type created specifically for this purpose: `clua`. This file type searches for files with extension `.dll` and `.so`. The `texmf.cnf` variable for this new file type is `CLUAINPUTS`, and by default it has this value:

```
CLUAINPUTS=.:$SELFAUTOLOC/lib/{$progname,$engine,}/lua//
```

This path a bit odd because it requires a TDS subtree below the binaries directory, but the architecture has to be in the path somewhere, and the simplest way to do that is to search below the binaries directory only.

In my case, the LuaTEX executable lives in `/opt/tex/texmf-linux/bin/` which replaces `$SELFAUTOLOC`,We will add the `luatex` path part as well as that is a nice thing to do (this replaces `$engine`), so the `lsqlite3.so` file should go here:

```
/opt/tex/texmf-linux/bin/lib/luatex/lua
```

Just copy the file there manually (again, you may need sudo rights).

To test, create a minimal input file `tests-sqlite3.tex` containing this:

```
\directlua {dofile('tests-sqlite3.lua'); }
\bye
```

and run it from that same directory:

```
[... lsqlite3_svn08]$ luatex tests-sqlite3.tex
```

You should get the same output again.

Taco Hoekwater
taco@luatex.org

Using ConTEXt with Databases

Accessing and typesetting information that is stored in databases is a common task. There are large-scale commercial solutions, and a number of database engines allow formatted output. In this article, I will show you one particular example: how I use ConTEXt MkIV to typeset material from a database. For my classes in Greek and Latin grammar, I have accumulated a large collection of exercises and examples from which I produce exercise sheets for my students. For a long time, I have relied on good old copy-and-paste to make new exercises and reuse some old material every year. But then I decided to do things in a more structured manner: I am in the process of putting all my examples into some sort of database from which the single exercise sheets will retrieve the exercises. This makes it easier to keep track of the material, make additions, and reuse elements in different ways without being too repetitive. In this article, I will demonstrate how ConTEXt can be applied to use such a database. There are two parts: in the first, you will see the new MkIV xml system in action; this new approach to processing xml from within ConTEXt makes it easy to access and manipulate parts of xml files. The second part will show a way to use sql databases as input for ConTEXt. I hope these examples can be useful for others who have similar needs.

The xml Database

The structure of our database is pretty simple: it has chapters covering single grammatical topics; every chapter has different examples. Every example has a unique identifier (expressed with an xml attribute `"id"`). There are two types of examples:

1. Normal examples have three elements: the “problem” (an English sentence or passage which is a translation of a Latin original), the “solution” (the Latin original), and the “origin” (the reference to the original, which is for my reference only and will not be typeset).
2. Some grammatical phenomena, however, can better be shown with Latin examples. In this case, we only have a Latin “problem” and an “origin.” These problems receive an identifier in the form of an xml attribute `type="latinonly"`.

Hence, a few examples from this database would look like this:

```
<examples>
  <chapter id="moods">
    <example id="deliberative1">
      <problem type="latinonly">
        quid ergo istius in iure dicundo libidinem et scelera demonstrem?
      </problem>
      <origin>
        Cicero, Verr. 2.39
      </origin>
    </example>
    <example id="indirect1">
      <problem>
        You do not see what he means.
```

```
      </problem>
      <solution>
        quid sentiat, non uidetis.
      </solution>
      <origin>
        Cicero, fin. 2.21
      </origin>
    </example>
    <example id="interdicere1">
      <problem>
        I have neither done it yet nor do I think it is forbidden to do it.
      </problem>
      <solution>
        id neque feci adhuc nec mihi tamen ne faciam interdictum puto.
      </solution>
      <origin>
        Cicero fin. 1.7
      </origin>
    </example>
  </chapter>
</examples>
```

The ConTEXt Environment

How can we make use of this xml database `examples.xml` in ConTEXt? We will use a ConTEXt environment to set up xml processing and format the output to our needs; this environment will be stored in a file `compositionxmlstyle.tex`. The first thing it does is define our environment:

```
\startenvironment compositionstyle

\stopenvironment
```

All the following lines go into this environment. We will now go through the TEX code step by step and see what it does.[1] We begin by simply loading our xml database:

```
\xmlloadonly{grammar}{examples.xml}{}
```

This simply makes the content of the database available to ConTEXt and it reserves the namespace `grammar` for this content.
We now have to process our database. There are two cases that we need to consider: the first is the "Problems" section. We want to be able to pick single problems, depending on their `id` attribute. Our first macro does just that: it extracts ("filters") a particular example:

```
\def\MyExample#1%
  {\xmlfilter{grammar}
   {/examples/chapter/example[@id=='#1']/command(xml:choose)}}
```

This macro is an instructive example of what the new MkIV xml mechanism can do.[2] As you see, it takes one argument, which it transfers to the command `\xmlfilter`. This command selects (or "filters") the content of our xml file (which is available under the name `grammar`). It traverses the structure of our xml file and picks the element `<example>` whose `id` corresponds to the argument of our macro; it then takes the content of the `<example>` element and transmits it to the command `xml:choose`.

Hence, this macro could be used in the form `\MyExample{deliberative1}` to pick the first example in our database.

```
\startxmlsetups xml:choose
  \doifelse {\xmlattribute{#1}{/problem}{type}} {latinonly}
    {\startitem[\xmlatt{#1}{id}]
      {\language[latin]\xmlstripped{#1}{problem}}
     \stopitem}
    {\startitem[\xmlatt{#1}{id}]
      \xmlstripped{#1}{problem}
     \stopitem}
\stopxmlsetups
```

When we "choose" our examples, we distinguish two cases: if the problem is of the `"latinonly"` type, it is a Latin phrase; otherwise, it is an English phrase. So we use the command `\doifelse` to distinguish between these two cases. This macro takes four arguments: the first two arguments are two strings that will be compared. If they are equal, the third argument will be executed; if they are not equal, the fourth. In our case, then:

- The command `\doifelse` looks at the attribute `type` of the sub-element `problem` of the current xml node (that's what `\xmlattribute{#1}{/problem}{type}` expands to).
- If the `type` attribute is equal to `"latinonly"`, the first branch is executed: we produce an `\item`; its reference (in square brackets) is the `id` attribute (`\startitem[\xmlatt{#1}{id}]`). For the text of the `\item`, we switch to the Latin language (to get proper hyphenation). `\xmlstripped` takes the value of the xml element and strips leading and trailing spaces, so the text in the `problem` subelement is typeset as the content of the item.
- If we have a "normal" example, with a `solution` subelement, we do the same thing, but we do not switch to Latin (because the sentence is English).

So now our "problems" are wrapped up as ConTEXt items, ready to be processed in a `\startitemize` environment. Next, we look at the solutions. We write a similar macro that filters examples; this time, it passes their content to a different command:

```
\def\MySolution#1%
 {\xmlfilter{grammar}
  {/examples/chapter/example[@id=='#1']/command(xml:solution)}}
```

This macro works exactly like the `\MyExample` macro. Next, we define the command `xml:solution`. Again, we will use ConTEXt's conditional mechanism: The `\doifnot` macro only processes examples that do *not* have a `problem` subelement of the `"latinonly"` type (remember, only these have a solution):

```
\startxmlsetups xml:solution
  \doifnot {\xmlattribute{#1}{/problem}{type}} {latinonly}
    {\SolutionMargin{\in[\xmlatt{#1}{id}]}
      {\language[latin]\xmlstripped{#1}{solution}}
       \par\blank[line]}
\stopxmlsetups
```

Every problem was converted into an item which had its `id` attribute as a reference. The second example from our database would thus be processed by ConTEXt as

```
\startitem[indirect1]
  You do not see what he means.
\stopitem
```

Our `xml:solution` command now picks up this reference (`\in[\xmlatt{#1}{id}]` will expand to `\in[indirect1]`) and wraps it into a ConTEXt macro `\SolutionMargin` (which we will define shortly). It then switches to Latin, gets rid of unwanted spaces, and typesets the text of the `solution` subelement, followed by a paragraph and an empty line.
Now, we prepare the look of our exercise sheets. We want problems and solutions to look exactly like the same. In both cases, we want the numbers to appear in the margin, in bold. So we first define `\SolutionMargin` as a margintext which will be typeset in the left margin:

```
\defineinmargin [SolutionMargin] [left] [normal] [style=bold]
```

Then, we define an itemgroup for our problems which will also display its numbering in the margin:

```
\defineitemgroup[MyExamples]
\setupitemgroup[MyExamples][n,inmargin]
\setupitemgroup[MyExamples][style=bold]
```

Now, the last macro we have to define; this is the one that we will really use in our document. Since we are lazy and want to type as few words as possible when we prepare our exercise sheets, this macro will do all the work for us:

```
\def\MyExercises[#1]%
  {\startsubsection[title=Problems]
     \startMyExamples
       \processcommalist[#1] \MyExample \par
     \stopMyExamples
   \stopsubsection
   \startsubsection[title=Solutions]
     \processcommalist[#1] \MySolution \par
   \stopsubsection}
```

Do you see what this macro does? It takes a comma-separated list as argument. It then starts a subsection (with title “Problems”), and within this subsection, it starts our itemgroup `\MyExamples`. It then processes our comma list and hands every argument over to the macro `\MyExample`, which in turn retrieves the examples from our xml database. Since, as you remember, this macro calls the helper command `xml:choose`, this will take the content of the problem and pass it to a `\startitem`, with reference `id`. Then, the macro inserts a new subsection (with title “Solutions”), processes our comma list again and typesets all the solutions as we defined in our `xml:solution` command, viz., with the reference to the problem in the margin. This guarantees that the numbering of problems and solutions will be consistent.

The User Interface

After all this hard work, we can finally reap the benefits: when we prepare our exercise sheets, we will only have to do a minimum of typing: we include our environment, we give some structure in the form of sections, and we include a comma-separated list of the examples from the xml database that we want typeset. All the rest is done by the macros we defined. So our document will look like this

```
\environment compositionstyle

\starttext

\startsection[title={Moods}]
  \MyExercises[deliberative1,indirect1,interdicere1]
\stopsection

\stoptext
```

This will typeset exercise sheets, complete with examples and solutions. But wait: what if you first want to give out exercises to the students without the solutions? Of course, you could postprocess the resulting pdf file and pick only the pages with the problems, but that would not be very elegant. A better solution is to build this capability right into our environment. We will use ConTEXt modes. We modify our main macro:

```
\def\MyExercises[#1]%
  {\startsubsection[title=Problems]
     \startMyExamples
       \processcommalist[#1] \MyExample \par
     \stopMyExamples
   \stopsubsection
   \startmode[solutions]
   \startsubsection[title=Solutions]
     \processcommalist[#1] \MySolution \par
   \stopsubsection
   \stopmode}
```

Now, the part of our macro which typesets the solutions will only be executed if the mode `solutions` is set. You can either insert a line `\enablemode[solutions]` into your file, or, even easier, you can set the mode when you call ConTEXt from the command line. When you typeset your file and want to have the solutions as well, the command is: `context --mode=solutions`; if you don't enable this mode, only the problems will be typeset.

Further Elements

As an example of what else we can do, I'll show you how you can handle more xml elements. Alas, students are not as fluent in Latin as they used to be a mere 450 years ago, so they sometimes need a little bit of help. How can this be integrated into our documents? First, let us look at the xml side. In order to give subtle hints, we just invent a new xml element `<hint>`; here is an example:

```
<examples>
  <chapter id="moods">
    <example id="interdicere1">
      <problem>
        I have neither done it yet nor do I think it is forbidden
        <hint>interdicere</hint> to do it.
      </problem>
      <solution>
        id neque feci adhuc nec mihi tamen ne faciam interdictum puto.
      </solution>
      <origin>
        Cicero fin. 1.7
```

```
      </origin>
    </example>
  </chapter>
</examples>
```

We want these hints typeset in the text of the problems, between square brackets, in italics. How do we do this? First, we have to "grab" these elements from our loaded xml file and connect them with a setup command:

```
\xmlgrab{grammar}{hint}{xml:hint}
```

Then, we define our setup command:

```
\startxmlsetups xml:hint
      \dontleavehmode[{\language[latin]\em \xmlflush{#1}}]
\stopxmlsetups
```

Which will take care of the `<hint>` elements, apply Latin hyphenation, and typeset them as we want them.

Other Databases: sql

If you did not like the preceding paragraphs, I have something else to offer: you may have been disappointed that the title of this article mentions "databases," yet all it talks about is xml. What if you want to use a "real" database format such as sql? ConTEXt MkIV can also cope with sql databases – or, to be more precise: Lua can, and so can ConTEXt with the luaTEX engine. I do not have the knowledge to make an in-depth comparison of xml with sql. If you search the web, you will see that people have (sometimes strong) preferences for one or the other. One advantage of xml is that it's stored in simple text files, in a human readable form; you are thus sure that your database will be usable for a long time. sql, on the other hand, has its advantages when it comes to speed (though the speed of the lookups is relatively negligible compared to the time it takes to typeset the document, so unless your database is really huge, there should not be much of a difference). One rule of thumb seems to be that xml is better for data which has a strong hierarchical structure, whereas relational databases (such as sql) are better for large sets of weakly structured data. When you look at our database of grammatical exercises, you will see that there is not much hierarchical structure; what we really need is to retrieve single examples by their `id`s, and this is something that sql is very good at. It is thus easy to think of a way to represent our grammatical exercises as an sql database. The table will need five columns (I abbreviate the text to make this representation easier to read):

```
id            | type     | problem     | solution     | origin
------------- | -------- | ----------- | ------------ | -------------------
deliberative1 | latin    | quid...     |              | Cicero, Verr. 2.39
indirect1     | both     | You do...   | quid...      | Cicero, fin. 2.21
interdicere1  | both     | I have...   | id neque...  | Cicero, fin. 1.7
```

It is easy to see how this is (almost) identical to our xml file. We now have a column `type` to distinguish between sets with and without a `solution`. The other columns correspond exactly to our xml tags. (We lose the information about the grammatical "chapter" to which every example belongs; if we wanted to, we could add another column to our database carrying that information).
This is not the place to give an introduction to sql, so I will be very brief here: I chose the `sqlite3` database management system because it is leightweight, open source,

available on many platforms, and does not rely on a server-client structure, hence it is well adapted for managing local databases.[3] `sqlite3` may already be available on your system, or else it can be installed quite easily. Creating such a database is easy. From the command line, we first create an empty database file:

```
sqlite3 grammar.db
SQLite version 3.7.3
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

We are now at the `sqlite` command prompt and can create our table:

```
CREATE TABLE examples (id TEXT PRIMARY KEY UNIQUE,
   ...> type TEXT,
   ...> problem TEXT,
   ...> solution TEXT,
   ...> origin TEXT);
```

This creates the structure (or "schema") for our table, and we can now insert our exercise problems; I give one example only:

```
INSERT INTO examples(id, type, problem, solution, origin)
   ...> VALUES('indirect1', 'both', 'You...', 'quid...', 'Cicero, fin. 2.21');
```

This will populate our table with our exercises, ready to be retrieved later.
How can we use an sql database in ConTEXt, then? Some years ago, Berend de Boer published a paper on this topic in the EuroTEX 2001 conference; of course, this applied to ConTEXt MkII.[4] He pointed out that it is fairly easy to insert xml tags or even ConTEXt commands into the output of an sql query. So one could massage the output and write it to a file, then call ConTEXt on that file. Berend proposed to do all this in a `perl` wrapper script. This would still be a viable route – but how much fun would that be? In MkIV, we can use a different approach: we can do the sql queries directly from within our document and typeset the results with ConTEXt. However, there are two caveats you will have to keep in mind when you read the remainder of this article:

1. I am not a database programmer by any means, I just impersonate one for this article. The code I will show you here does work for me, but it may be quite naive or unsophisticated – you are welcome to improve it!
2. Unfortunately, what I have written in the first paragraph of this section is not quite literally true: Lua can indeed deal with sql, but it needs additional modules to do so. Unfortunately, I found the situation a bit confusing: there are (at least) three different modules that allow Lua to work with sqlite databases.[5] Even after doing some research on the web, I could not quite figure out in which relation these modules are – they are quite similar in their basic approach, but differ in many aspects of the user interface.[6] For this article, I use the Lua module `LUASQLite3`,[7] which appears to be the only one which is still actively maintained. In another article in this issue, Taco Hoekwater explains how to install a Lua module so that luaTEX and ConTEXt can actually use it.

With all this understood, let us set our specifications for what we want to achieve, then. We want to keep exactly the same user interface in our ConTEXt file as in the first part of this article when we were dealing with xml; i.e., we still want to use our

macro \MyExercises[] with a comma list of examples; and we want to be able to use modes to have our solutions typeset or not. Our aim is to produce a universal macro that can be driven either by an sql database or by an xml file, without the user having to worry about it.

So let us roll up our sleeves: our database grammar.db is in place; its table examples is populated with our exercises; our luatex binary is able to find and use the lsqlite3 module. What would our new environment look like? Much of what we will do now will be done in Lua, so we begin by writing our Lua code. I find it convenient to write and test my Lua code first and wrap it into the proper \startluacode \stopluacode environment later, but this is just a habit.

We could, of course, reuse some of the code we have written for handling the xml file, especially the part where we used the neat \processcommalist macro, but since we will be writing a Lua function anyway, I found it interesting to see how much of this could be done in Lua. As you will see, it is possible to write the cimplet set of processing and typesetting commands in Lua. This may sometimes appear a bit convoluted, but it may inspire you when you want to write your own Lua functions, so here we go: our Lua function (which will later be wrapped into a \ctxlua macro) will take as its argument a comma-separated list of values. So the first thing we have to do is split this list into its elements and make a table out of them; we use the lpeg library for this.[8] In the following code, the variable keywordlist designates what will be the user input when we define our ConTEXt macro.

```
userdata = userdata or { }
userdata.sql = userdata.sql or { }
userdata.sql.sep = lpeg.P(",")
userdata.sql.mywords = lpeg.C(( 1 - userdata.sql.sep )^0)
userdata.sql.p = lpeg.Ct(userdata.sql.mywords *
                 (userdata.sql.sep * userdata.sql.mywords)^0)
userdata.sql.mytable = lpeg.match(userdata.sql.p, keywordlist)
```

This creates a Lua table userdata.sql.mytable with all the keywords in it. We will later use this table to retrieve the single problems from our database. But let us first look at the way we will be accessing the database itself.

The Lua function proper will query the database for entries whose id corresponds to this element. Here is how this can be achieved:

```
require("lsqlite3")

userdata.sql.mygrammar = assert (sqlite3.open("grammar.db"))

function userdata.sql.getproblem(myid)
  userdata.sql.myquery =
    userdata.sql.mygrammar:prepare("SELECT problem FROM examples WHERE id = ?")
  userdata.sql.myquery:bind_values(myid)
  userdata.sql.myproblem  = userdata.sql.myquery:get_value(0)
  userdata.sql.myquery:step()
  userdata.sql.myquery:finalize()
end
```

Let us have a brief look at this code. This is the part where Lua interacts with our database. The first thing we have to do is load ("require") the lsqlite3 module. We use it to open our database and give a symbolic name to the resulting Lua structure. We then run an sql SELECT query on the table examples in this database. This query creates an object, to which we again assign a handle, userdata.sql.myquery. The interesting part here is the end of the query: in WHERE id = ?, the question mark

is a placeholder which we then "bind" to the argument of our Lua function, `myid`. Our query selects the column `problem` from the database which is captured in the function call `userdata.sql.myquery:get_value(0)` (if we wanted to retrieve `n` more columns, these would be captured as `userdata.sql.myquery:get_value(0+n)`). We assign a Lua variable to this result. The function call `userdata.sql.myquery:step()` will actually apply our query to the next row of the database; the query is closed with `userdata.sql.myquery:finalize()`.
So all we have to do now is write a loop which will take the single elements of our `userdata.sql.mytable`, make sure to get rid of all whitespace which users may put into this comma list, pass the single values on to the `userdata.sql.myquery` function, and then do something with the results we receive. Since we want the results to be the same as with the xml example, we reuse the setups for our item lists and our margin numbers:

```
\defineinmargin [SolutionMargin] [left] [normal] [style=bold]

\defineitemgroup[MyExamples]
\setupitemgroup[MyExamples][n,inmargin]
\setupitemgroup[MyExamples][style=bold]
```

And this is how we will use these definitions in our Lua loop:

```
context.startsubsection( { "title=Problems" } )
context.startMyExamples()

for k, myid in ipairs(userdata.sql.mytable) do
  myid = myid:gsub(" ", "")
  userdata.sql.myquery =
    userdata.sql.mygrammar:prepare("SELECT problem, type
                              FROM examples WHERE id = ?")
  userdata.sql.myquery:bind_values(myid)
  userdata.sql.myquery:step()
  userdata.sql.myproblem = userdata.sql.myquery:get_value(0)
  userdata.sql.mytype = userdata.sql.myquery:get_value(1)
  userdata.sql.myquery:finalize()
    context.startitem( { myid } )
      if userdata.sql.mytype == "latin" then
        context.bgroup()
        context.language( { "latin" } )
        context.delayed(userdata.sql.myproblem)
        context.egroup()
      else
        context(myproblem)
      end
    context.stopitem()
end
context.stopMyExamples()
context.stopsubsection()
```

What you see here is ConTEXt code written in Lua. Every ConTEXt command has a corresponding Lua equivalent. If you define an environment `\MyExamples`, the Lua function call `context.startMyExamples()` is equivalent to `\startMyExamples`.[9] As I said before, we could have done most of this in ConTEXt itself; I just wanted to demonstrate this Lua interface here.

But there is more! Remember, we also wanted to typeset the solutions for problems that were of `type latinonly` if the mode `solutions` was set. Here is how we can do this in Lua:

```
if tex.modes["solutions"] then
  context.startsubsection( { "title=Solutions" } )
  for k, myid in ipairs(userdata.sql.mytable) do
    myid = myid:gsub(" ", "")
    local userdata.sql.myquery = userdata.sql.mygrammar:prepare
            ("SELECT type, solution FROM examples WHERE id = ?")
    userdata.sql.myquery:bind_values(myid)
    userdata.sql.myquery:step()
    userdata.sql.mytype = userdata.sql.myquery:get_value(0)
    userdata.sql.mysolution = userdata.sql.myquery:get_value(1)
    userdata.sql.myquery:finalize()
    if userdata.sql.mytype == "both" then
      context.SolutionMargin(context.delayed["in"]( { myid } ) )
      context.bgroup()
      context.language( { "latin" }, context.delayed(mysolution) )
      context.egroup()
      context.par()
    end
  end
  context.stopsubsection()
end
```

As you can see, we have to query the database a second time, to retrieve the solutions. This time, we also need to retrieve the `type` column since only database entries with `type both` do, in fact, have a solution, and we need to test this (otherwise, Lua will complain because the instruction `context(userdata.sql.mysolution)` may result in an empty argument). You will find it easy to recognize the other elements which we have already done in the first part: we test whether the mode `solutions` is enabled; if it is, we further test whether the `type` of the entry is `both`; if it is, we typeset it, with its `id` as a reference to the item in the problem list.
So all we need to do now is wrap the entire Lua code in the proper environment (I give just the beginning and the end) and define the command that we will use in our file:

```
\startluacode
require("lsqlite3")
userdata.sql.sep = lpeg.P(",")
function userdata.sql.getexample(keywordlist)
...
end
\stopluacode
\def\MyExercises[#1]%
  {\ctxlua{userdata.sql.getexample("#1")}}
```

As you see, in our sql environment, the macro `\MyExercises` passes its argument over to our Lua function `getexample`, which will in turn split it into its single keywords, query the database for them, and finally typeset the corresponding problems and solutions. So we have achieved exactly what we wished: we have defined the same macro which will now retrieve our exercises from an sql database!

What about our special <hint> element? If you remember, we wanted these hints typeset within brackets, and in italics. How can we integrate this into our sql approach? One solution would be to write the ConTEXt code which you want evaluated directly into the entry in the database, so the problem column of our example would look like this:

```
... it is forbidden [{\language[latin]\em interdicere}] to do it.
```

If you are certain that you will never use your database with any other (necessarily inferior) tools than ConTEXt, this would be a possible way, but it is not very elegant. Better to keep the database as generic as possible and massage the data at the ConTEXt end. So we have to think of a delimiter for our hints – this must be a pair of characters that you will not use in any other way. In our case, square brackets are used for nothing else but to include such hints, so our database simply contains:

```
... it is forbidden [interdicere] to do it.
```

When we retrieve the problems in our Lua code, we simply replace these brackets with the code:

```
function userdata.sql.debracket(s)
   p = string.sub(s,2,-2)
   return p
end

userdata.sql.myproblem = userdata.sql.myproblem:gsub
  ("(%b[])", function(t) return "[{\\language[latin]\em"
   .. userdata.sql.debrac(t) .. "}]"; end)
```

This operation on the string with Lua's gsub command will simply replace all strings within balanced brackets (such as [interdicere]) in the output of our query with [{\language[latin]\em interdicere}].

Conclusion

The actual output of our exercise sheets doesn't look very exciting yet (actually, “Problems” and “Solutions” will be typeset on two different pages, but here I have indicated the page break by a simple line).

Moods

Problems

1 quid ergo istius in iure dicundo libidinem et scelera demonstrem?
2 You do not see what he means.
3 I have neither done it yet nor do I think it is forbidden [*interdicere*] to do it.

Solutions

2 quid sentiat, non uidetis.
3 id neque feci adhuc nec mihi tamen ne faciam interdictum puto.

But it is easy to add bells and whistles, color, different fonts and sizes, etc. It's all a matter of adapting settings in your environment. The example I have shown here may be a bit specialized, but it should allow you to appreciate the simplicity of the underlying mechanism.

Footnotes

1. As always, I would not have been able to figure all this out myself. I gratefully acknowledge the help of the ConTEXt community on the mailing list; in particular, Aditya Mahajan and Peter Münster have provided valuable help. And, as always, none of this would have been possible without Hans Hagen's kind support.
2. If you are curious and want to know more about xml in ConTEXt MkIV, you should have a look at the manual which can be downloaded at http://www.pragma-ade.com/general/manuals/xml-mkiv.pdf.
3. For more information, point your browser at http://sqlite.org/.
4. The paper is available at http://www.ntg.nl/eurotex/deboer.pdf.
5. See the somewhat terse wiki page at http://lua-users.org/wiki/LuaSqlite.
6. It is somewhat reassuring to see that other users feel confused, too; see the questions at http://lua-users.org/lists/lua-l/2009-03/msg00405.html.
7. See http://luaforge.net/projects/luasqlite/.
8. The following code is adapted from the `lpeg` website at http://www.inf.puc-rio.br/~roberto/lpeg/lpeg.html#ex.
9. For more information, see Hans Hagen's article "Typesetting in Lua using LuaTEX" in the previous issue of *MAPS* and the manual at http://www.pragma-ade.com/general/manuals/cld-mkiv.pdf.

Thomas A. Schmitz

Gabo's Torsion

—and some more—

Abstract
Gabo's Torsion is emulated in EPSF, Encapsulated PostScript File format. Gabo's constructive art, Math, Computer Graphics and the use of PostScript are touched upon. Whether PostScript is a suitable language for projection and drawing 3D objects on paper is experienced. An introduction to PostScript aimed at EPSF use, in a nutshell, is included. How to obtain cropped pictures along with the conversion to .pdf is mentioned. An interesting observation is made: Bézier cubics, specified by begin point, the control points and the end point, are invariant under (oblique parallel) projection, which allows to project B-cubics efficiently. The efficient projection of (approximated) circles and ellipses has been addressed. For the evaluation of B-cubics de Casteljau's algorithm is used. Emulations in EPSF of Gabo's Linear Construction in Space No 1 and 2, of one of his Spheric Themes, and his Linear Construction Suspended, are also included. For the MetaFont aficionados my interactive version of old is also included.

Keywords
2.5D, art, AFII, ASCII, astroid, Bernstein polynomials, BoundingBox, Bézier cubic, de Casteljau algorithm, ConTEXt, cropping on-the-fly, EPSF, Gabo, hyperboloid, METAFONT, MetaPost, minimal encapsulated PostScript, plain TeX, projection, PSlib, stringed surface, TEXworks

Abbreviations
3D (3 Dimensional), ASCII (American Standard Code for Information Interchange), AFII (Association for Font Information Interchange), ATN (Adobe Technical Note), BB (BoundingBox), CAD (Computer Aided Design), DVD (Digital Versatile Disk), EPSF (Encapsulated PostScript File format), GS (GhostScript), IDE (Integrated Development Environment), IMHO (In My Honest Opinion), LL (LanguageLevel), LRM (Language Reference Manual), PDF (Portable Document Format), PS (PostScript), PSlib (my PostScript library), MF (METAFONT), MP (MetaPost), MS (MicroSoft), PC (Personal Computer), RF (Reference Manual), US (User Space), WWW (WorldWide Web), XPS (XML Paper Specification of MS, a functional subset of PDF), XML (eXtensible Markup Language).

Introduction

Nearly 50 years ago, while on a physics students excursion in the UK, I visited the Tate Gallery. I was captivated by LINEAR CONSTRUCTION IN SPACE No 2 by Naum Gabo.[1] Much later, in the mid 90s, I emulated this object in MetaFont, and Jos Winnink processed my adapted MetaFont code for MetaPost. Earlier, in the 80s, I made a perspex emulation and also did Gabo's TORSION in triplex. The tedious stringing was done together with my youngest daughter. On my WWW of old I even had a quasi-animation of LINEAR CONSTRUCTION IN SPACE No 2. For the EuroTEX&ConTEXt 2009 Jos and I had to repeat the processing of the .mp files, because the .eps files were lost, after 15 years.

Lauwerier(1987) treats the projection of polyhedra, crystals, toroid, sphere with meridians, and ... in BASIC. The present work goes beyond Lauwerier in the sense that projection of planar curves, with circles and ellipses as special cases, is done via B-cubics, in time-proven EPSF.

Of late, I had the inspiration to emulate Gabo's TORSION, see below, in EPSF directly.

I have no access to CAD/CAM software nor Mathematica nor ... to emulate 3D, but ... happily

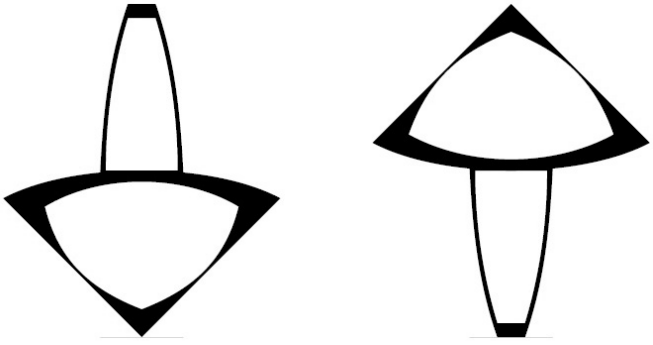
EPSF is a TEXie's graphics companion

For completeness I have included in the Appendices emulations in EPSF of Gabo's LINEAR CONSTRUCTIONS IN SPACE No 1 and 2, as well as one of his `Spheric Themes`, and on the nick LINEAR CONSTRUCTION SUSPENDED.

Analysis of the Torsion object

The construction, a stringed metal frame, consists essentially of 2 identical rectangular isosceles triangles, with a curved hypotenuse and curved inner sides, on top of each other, with the upper triangle turned upside down and rotated over 90°.

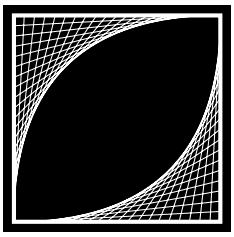
Frame. In the accompanying picture at the left the lower triangle with on top what I call a cap. At the right the upper triangle with the mirrored cap under it, which I call a cup.



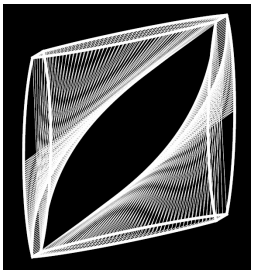
Stringing. The inner curves of the lower triangle are connected by lines to the inner curves of the upper triangle, the strings of the object. Cap and cup have horizontal strings. The stringing yields what I call stringed surfaces.

Stringed surfaces. intrigue me because the impression of a curved surface is obtained from straight lines, which only require 1D information: the boundaries of the frame.

My youngest daughter at primary school in the 70s made already a sort of Gabo in 2D, branches of an astroid, which I emulated.



← 2D stringed surface
with spurious envelope
Gabo's emulated 2.5D →
LINEAR CONSTRUCTION No 1



As appetizer of the use of EPSF the program of the `2D stringed surface` is given, which reflects the essentials and structure of the codes for the emulations of the TORSION, the LINEAR CONSTRUCTION, the SPERICAL THEME, and LINEAR CONSTRUCTION SUSPENDED objects.

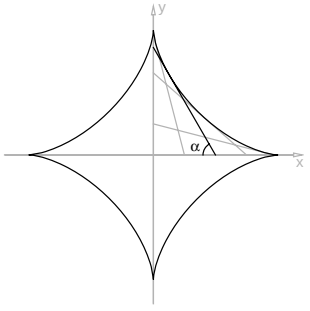
```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -55 -55 55 55
%%BeginSetup
%%EndSetup
%%Title: 2D stringed surface (envelopes are branches of astroid)
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%CreationDate: Februari 2011
%%EndComments
%%BeginProlog
/reversevideo{-55 -55 110 110 rectfill}def %(Black) Background; Mimics BoundingBox
%
/framestroke{-50 -50 100 100 rectstroke}def
%
/dostringing{0.05 .05 1.01{%for
    /incr exch s mul 2 mul def
    s neg           s neg incr add     moveto
    s neg incr add  s                  lineto
    s neg incr add  s neg              moveto
    s               s neg incr add     lineto
}for}bind def %end dostringing
%%EndProlog
%
%---Program--- the script
%
reversevideo 1 setgray      %black background and white lines, further on
1  setlinejoin 1.415 setmiterlimit 1 setlinecap
2  setlinewidth framestroke %paint frame to the current page
.1 setlinewidth dostringing stroke %paint strings to the current page
showpage %send the current page to the raster device, printer...
%%EOF
```

Note. Because I made use of `rectstroke` in the `framestroke` procedure the `setlinewidth` must precede `framestroke`. In all emulation programs rounded line joins and line caps were asked for and spikes were suppressed by the settings as given in the example program.

Equation of the envelope? Let us spend a little time on refreshing our analytic geometry.

> If a family of curves $f(x,y,\alpha) = 0$, parameterized by α, is tangent to a plane curve E then the curve E is the envelope of the family of curves.

An envelope E is characterized by $\begin{pmatrix} f(x,y,\alpha) \\ f_\alpha(x,y,\alpha) \end{pmatrix} = 0$. Suppose our family of curves are the straight lines of unit length intercepted by the x and y axis, which make an angle α with the x axis: $\frac{x}{\cos\alpha} + \frac{y}{\sin\alpha} = 1$.



$$\begin{pmatrix} f(x,y,\alpha) \\ f_\alpha(x,y,\alpha) \end{pmatrix} = 0 \quad \rightarrow$$

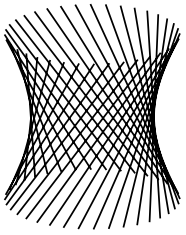
$$\begin{pmatrix} \frac{1}{\cos\alpha} & \frac{1}{\sin\alpha} \\ \frac{\sin\alpha}{\cos^2\alpha} & -\frac{\cos\alpha}{\sin^2\alpha} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \rightarrow$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos^3\alpha \\ \sin^3\alpha \end{pmatrix}$$

Elimination $\cos\alpha$ and $\sin\alpha$ in the equation for the line $\frac{x}{\cos\alpha} + \frac{y}{\sin\alpha} = 1$ yields $x^{2/3} + y^{2/3} = 1$, an astroid, also known as hypocycloid with 4 branches.[2]

My daughter's construction was made differently, not based on a 'unit length intercepted by the x and y axis'. But ... a similar reasoning as above yields the equation $\sqrt{x} + \sqrt{y} = 1$; I'll call her construction: astroid alike.

A stringed surface with known equation. If we rotate a line l around the z axis we obtain a stringed surface. It is a stringed surface because another way to construct it, is starting from 2 horizontal circular frames and connect points of the circumferences by straight lines.



Let l be given by $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1-t \\ t \\ 2t-1 \end{pmatrix} \quad t \in [0,1]$

Squaring and eliminating t yields

$2(x^2 + y^2) - z^2 = 1$, a hyperboloid

Courtesy: Lauwerier(1987)

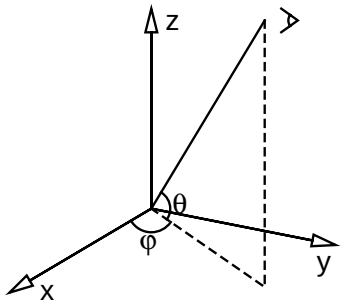
Gabo's stringed surfaces are too complex to be described in equations.

Projection

At high school I learned in the stereometry class to draw projections in the spirit of the drawing in the MetaFont book p113 ex13.7, and as detailed in Lauwerier(1987) Ch3.

Nowadays, because of the ubiquitous PCs, emulation of a 3D object is generally done by (oblique parallel) projection, i.e. the object is viewed under (φ, θ), the azimuth and inclination (Azimuth φ is the angle between the *x* axis and the view direction. Inclination θ is the angle between the view direction and the *xy* plane. In the picture φ and θ are positive.)

Moving the view direction over φ is the same as the object rotating over $-\varphi$, be aware of confusing the two. Trivial, ubiquitous examples of projection are photographs. Lauwerier makes the comparison: the shadow of a frame of wire, as e.g. in a sun dial. The projection plane is orthogonal to the view direction.



right-screw
coordinate system
with view direction φ,θ
in the main octant

In order to project a 3D curve the brute force method is to sample the curve and connect the projected samples by `linetos`. This is done in the SPHERICAL THEME emulation, see Appendix 5. In the LINEAR CONSTRUCTION IN SPACE No 2 emulation, see Appendix 4, the symmetry of the frame is used. In general when we approximate a curve by B-cubics we can project more efficiently.

Properties of (oblique parallel) projection

- straight lines remain straight lines (2 points on the line are sufficient for projecting the line) with preserved ratios
- circles become ellipses (for efficient projection see Appendix 1)
- Bézier cubics (as used in PostScript) are 'invariant' (see Appendix 1)
- angles are **not** preserved, i.e. parallel projection is not conformal
- shape of curves are in general **not** preserved; projections are done by projections of B-cubic approximations or by projection of sample points.

For drawing 3D objects in PostScript I discern the following spaces

- PostScript's Device Space and User Space
- the mathematical 3D User Space, the data.

3D data are projected on 2D PostScript US with the paths constructed in the projection plane. The result I call a 2.5D image.

The projection formula (see Appendix 0 or Lauwerier(1987) p48 (3.7)) reads

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\sin\phi & \cos\phi & 0 \\ -\cos\phi\sin\theta & -\sin\phi\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

with $(x, y, z) \in \mathbb{R}^3, (u, v) \in$ Projection plane.

For the accompanying illustration of the coordinate system the view direction $\phi = 35^\circ$ $\theta = 20^\circ$ was used.

A feeling for the formula can be obtained by imagining special cases, such as: $\phi = 0$, $\theta = 0$: view of yz plane, i.e. (1,2,3) → (u,v) = (2,3), etc.

On the occasion of the joint EuroTEX&ConTEXt 2009, I programmed the projection in PostScript as procedure with name `ptp`, mnemonics pointtopair.

```
/ptp{% point x y z ==> u v, the projected point
       % example of use: /pair { x y z  ptp } def  %with deferred execution
       % parameters: phi, theta: the viewing angles azimuth and inclination
       % xyz coordinate axes: x to you, y right, z up, right-screw
   ptpdict%push ptpdict on the operand stack
   begin  %move the ptpdict dictionary from the operand stack to the d-stack
```

```
   /z exch def/y exch def/x exch def
   x phi sin mul neg              y phi cos mul add
   x phi cos mul theta sin mul neg y phi sin mul theta sin mul sub
                                                    z theta cos mul add
   end% pop the current dictionary, ptpdict, from the d-stack
} bind def
/ptpdict 3 dict def % create dictionary with name ptpdict
```

The use of `ptpdict` is paramount with such common (local) variable names, like `x`, `y` and `z`. The 'global' parameters ϕ, θ must be initialized in the dictionary, to avoid mix up and let them behave as locals to the procedure `ptp`.

Hidden lines.
A side effect in projection are lines which should not be visible, the so called hidden lines. I use one colour throughout with the pleasing result that I don't have to worry about hidden lines. In Gabo's transparent perspex objects everything is visible.

In the current object one might handle hidden lines by a judiciously chosen printing order of the elements, with the triangles split longitudinally along the axis. This works for $\phi \in [0{,}90°]$, $\theta \in [-90°{,}90°]$.

Emulation

My first emulation of TORSION was in 3D with the frame of triplex in the late 80s. Two sides are broken after so many years.

My emulation of LINEAR CONSTRUCTION IN SPACE No 2, in perspex stringed by nylon filament, has not stand time, which also has happened to some of Gabo's objects. The MF emulation and the reverse video picture is published in MAPS 16-28; the picture is also included in the LaTeX Graphics companion. I improved the MetaFont emulation on the occasion of the joint EuroTeX&ConTeXt 2009, by thinner strings and showing all the strings, also the no longer 'blurring' hidden ones.

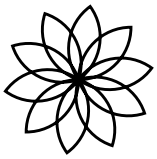
Why in PostScript?
The use of a programming language has two aspects: the richness and power of the language proper and the ease of working in an IDE.

PostScript language. The language was designed by Adobe, and introduced in 1985, as a device-independent page description language with powerful graphics capabilities, ≈ 400 operators — i.e. built-in procedures of the system dictionary — in PostScript level 1, with substantial more in PS level 3. The extensive graphics capabilities

are embedded in the framework of a general-purpose programming language. The language includes a conventional set of data types, such as numbers, booleans, arrays and strings; control primitives, such as conditionals, loops and procedures; and some unusual features, such as dictionaries, next to higher-level structures, such as patterns and forms. These features enable application programmers to define higher-level operations that closely match the needs of the application and then to generate commands that invoke those higher-level operations. The LRM version 3 is the defining document for the syntax and semantics of the language, the imaging model, and the effects of the graphics operators.

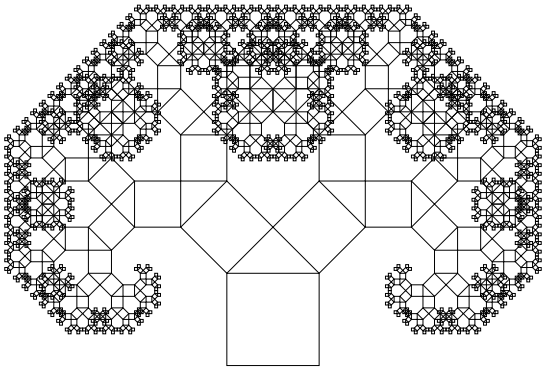
Powerful concepts:

- A user space which can be altered: 'the coordinate system's origin may be *translated*, moved to any point in user space; the axes may be *rotated* to any orientation; the axes may be *scaled* to any degree desired; the scaling may be different in the *x* and *y* directions.' Reflection and skewing are also supported. My favourite illustrations of the US concept are a stylistic flower and the recursive programming of the Pythagoras tree, which is all about drawing a square in repeatedly transformed US.



← Stylistic Flower

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -26 -26 26 26
%%BeginSetup
%%EndSetup
/r 18 def
10 {0 r r 270 360 arc
    r 0 r  90 180 arc
    36 rotate} bind repeat
stroke showpage
```



Pythagoras Tree →
BachoTEX 2011 pearl

Another nice example of the usefulness of transforming US is to create a path of an ellipse by the use of the `arc` operator, for example `1 2 scale 0 0 25 0 360 arc` (Courtesy the Blue Book, but ... be aware of the fact that the pen width has been scaled too).
To translate the centre of the coordinate system, default in the left lower corner of the current page, was the first thing I used to do. No longer necessary for my stand-alone illustrations in an EPSF program, which begin with the 4 lines as given in the stylistic flower example.

- The colour spaces, which notion was introduced in PostScript level 2 of 1991, and elaborated upon in 1997 in PostScript level 3, with among others much more efficient shading functionality. In PostScript level 1 there were already the concepts colour mode and half-tones, with operators `setrgbcolor` and `setgray`, which were generalized in level 2 into `setcolorspace` with `setcolor`. The chapter headings of the level 1 Red and Blue Books reflect the gradients, or smooth shading, functionality.

Inspired by The Green Book p139
Much can be learned from this example
which was state of the art in 1985
Note, however that
`%%BeginSetup`
`%%EndSetup`
are not necessary???
Level 3 features rich colour shading

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 200 36
/DataString 256 string def
/oshow {true charpath stroke} def
/H20 {/Helvetica-Bold 20 selectfont} def
/H8 {/Helvetica 8 selectfont} def
%%EndProlog
0 0 moveto
gsave 175 36 scale
0 1 127 {DataString exch dup 128 add put}bind for
128 1 8 [128 0 0 1 0 0] {DataString} image
grestore
0 0 200 36 rectstroke 1 setgray
H20 95 14 moveto     (PostScript) show 0 setgray
    95 14 moveto     (PostScript) oshow
H8  95  4 moveto 2 0 (Language)   ashow
showpage
%%EOF
```

The number of pages of the PostScript level 3 LRM has increased to 912p, while the number of pages of the level 1 LRM is 321p. See Appendix A of the LRM 3 for the ways the PostScript language has been extended with new operators and other features over time. The language versions are upward compatible.

- Handy and useful optimizations, such as `rectstroke` `rectfill` `userpath` `selectfont` ...
  but ... don't use `store` in library procedures instead of `def`, with variables which **are intended** to have a local scope. Use a dictionary local to the library procedure for variables used more than once.
- The graphics state, with commonly used operators `gsave` `grestore` ... .
- The current page, the abstract page, the raison d'être concept behind PostScript, to be rendered by the interpreter in the rendering device (printer, screen, ...), commanded by the `showpage` operator.
- Variability of fonts by the font transformation matrix as argument of `makefont`, with scalability, mirrored fonts (as in XƎTEX) smallcaps, and outlines as obvious examples.
- Stacks `operand dictionary graphics state execution`
- Immediately evaluated names, i.e. `//name` is immediately replaced by its value; useful for named constants.
- `bind` operator which looks up values of the **operator** names in the procedure operand and replaces these by their values, the so-called early binding, and returns the modified procedure. It enhances efficiency and robustness against (unintended) change of used operators, especially in library procedures. Optimize loop bodies too by `bind`.
- Idiom recognition feature of Level 3. A functionality added to the bind operator, which can be switched on/off by setting the user parameter `IdiomRecognition`. Bind can find and replace commonly occurring operators, called *idioms* by operators of higher quality and/or better performance. For example PostScript level 2 shading operators are replaced by PostScript level 3 improvements, silently behind the scenes, with new snazzy codes.
- `execform` for repeatedly placing a graphic, e.g. a logo, a fill-in form, ... efficiently (since level 2).
- Operator groups, with a few operators named from each group:
  — operand stack `pop exch dup ...`
  — arithmetic and math `add div ... srand`
  — array `array ... getinterval`[3]
  — dictionary `dict ...dictstack`
  — string `string ... run ... token`

- relation, boolean, and bitwise `eq ... bitshift`
- type, attribute, and conversion `type ... cvs`
- file `file = == ... echo`
- virtual memory `save restore ...`
- miscellaneous `bind null usertime version`
- graphics state `gsave grestore ... currenttransfer`
- coordinate system and matrix `matrix ... invertmatrix`
- path constructing `newpath`, `moveto lineto curveto arcto ... clip eoclip ... charpath ...`
  Only one path is possible, although by juggling with several graphics states one may maintain several collateral paths
- painting to the current page of
  paths `stroke` paints lines `fill eofill ...` fills an area
  strings `show ...`
  a sampled image `image shshow ...`
- device setup and output `showpage ...`
- character and fonts `findfont scalefont setfont` (or level 2 onward optimized concatination of the 3 into `selectfont`) `makefont` (transforms more general than `scalefont`) `... kshow ... cshow ...`
- font cache `setcharwidth ...`
- errors `dictfull ...VMerror`.

Overwhelming, isn't it?

Let us pick out a few, which I use most of the time, apart from the arithmetic, math and relation operators, whose use we are already familiar with from our favourite programming language.

`def`	associates names with procedures or values available on the operand stack, and stores the associated pair in the current dictionary
`moveto`	creates the starting point of an (internal) path
`stroke`	and ilks, to paint the path to the current page
`image`	to render the (bitmap) image onto the current page
`gsave`	pushes the current graphics state on the gs-stack and creates a new current one
`grestore`	pops the (top) graphics state off the gs-stack and makes it the current graphics state, en passant obseleting the graphics state in use
`makefont`	is used by Don Lancaster (and undoubtedly by others) for creating a variety of fonts from the available ones. He calls his collection 'fonts for free'. I love his embossed variant
`kshow`	I used kerning show in my BachoTEX 2010 pearl for the typesetting of π-decimals along a spiral
`forall`	handy for creating concise code, also used in my BachoTEX 2010 pearl.

The language is stable and flexible, also called extensible, meaning one can add procedures. It is the industry page description standard language. Programs are interpreted, line by line, not compiled, which at the time was important because of small memories. It is maintained by Adobe — the stewards of PostScript— and already with us for more than 25 years! Interpreters are generally provided by 3rd parties, especially the interpreters which come with your printer.

The Red, Green and Blue Books — Reference Manual, PostScript Language Program Design, respectively Tutorial and Cookbook — are published by Addison-Wesley and also available for free on the WWW, even the level 1 and 2 (774p) LRM's. The Blue Book, which was aimed at to set a standard for effective PostScript programming, contains examples of procedures, such as `oshow outsidecircletext insidecircletext pathtext printposter DrawPieChart centerdash smallcaps` and `arrow`, to

name but a few.[4] The Red Book is indispensable. The Green Book is about software engineering in PostScript, not only to get the programs to work, but to create correct, readable, efficient, maintainable and robust PostScript programs. The underlying goal is to develop a printer driver. There is also an Adobe White Book about Type 1 fonts, also for free on the WWW. Mnemonics: the RGB-collection of Adobe :-). PostScript programs, in ASCII, are generally generated by programs, hardly self-written. They facilitate exchange of (stand alone EPSF) picture descriptions, and of course (the pages of) a complete publication. The structuring conventions of Appendix C of the Red Book level 1 have grown out into an Adobe Technical note #5001, 109p. Nowadays, illustrations are easily exchanged in `.pdf`, and everybody can view them because of the ubiquitous, free Acrobat reader. The Mathematica reader is also freely available, and facilitates the exchange of Mathematica `notebooks`. The exchange of ASCII PostScript is useful.

Although a powerful graphical language, PostScript is considered low-levelish by the TEX community at large. They favour John Hobby's preprocessor MP, Knuth included, with exceptions: Taco Hoekwater, me ... . Taco includes PostScript on-the-fly in `escrito`, his PostScript compatible interpreter in Lua. He is also on the MetaPost maintenance team and works on extensions of MetaPost. At BachoTEX 2010 he announced the release of MetaPost 2.000.

> 'PostScript is underestimated and underused,' to quote Taco Hoekwater.
> 'I agree with him ... I'm not saying he is right ;-),' well ... he is.

IMHO, a little bit of PostScript does not harm. On the contrary, you will benefit from the general-purpose programming language framework, the imaging model, or you may extend your knowledge about the interactive system for controlling raster output devices, but ... self-study is dangerous. What we need is a teacher à la John Deubert who thoroughly understands the PostScript concepts. John in his Acumen Journal pays attention to among others the relation of PostScript to `PDF` and `XPS`, and gives many nice, good and useful examples, clearly explained line by line.

From the LRM

> ' PDF lacks the general-purpose programming language framework of the PostScript language. A PDF document is a static data structure that is designed for efficient random access and includes navigational information suitable for interactive viewing.'

Finally, and in contrast with TEX, a mnemonic

All what you type in PostScript are

- comments after %
- numbers
- operators
- names to be looked up in the dictionaries, and at last
- strings which contain text.

From the LRM

> ' The interpreter manipulates entities called PostScript objects. Some objects are data, such as numbers, boolean values, strings, and arrays. Other objects are elements of programs to be executed, such as names, operators, and procedures.'

In TEX the source is the text of the publication, interspersed with as few markup commands as possible, at least that is what Knuth aims at in his minimal markup style, which I love and practice too.

So ...
add def ... are names to be looked up in the dictionaries
(< / [... are operators:

(starts a string, where all is interpreted as text, with \ as escape character; a TEXies niche
< starts a hexadecimal string, consisting of the 'digits' in the hexadecimal system 0,1, ... 9, a, b, c, d, e, f, commonly used in the executable array, ie, procedure, as argument for the image operator
/ starts a literal name
[starts an array, which may contain heterogeneous elements, in contrast with other languages.

Be aware of the difference between name and /name. The first is a name to be looked up in the dictionaries, while the slash in the second starts a *literal* name, which is only pushed on the operand stack, without execution. Unlike other programming languages such as PASCAL there are no reserved words.

Comments start with %. Comments are used for structuring. Special comments are

%! the start of a PostScript program with %!PS-Adobe-3.0 EPSF-3.0 the complete line for illustrations to be encapsulated
%% at the beginning of a line starts structural information about the PostScript program, as explained in Appendix C of the LRM version 1, or see ATN #5002; syntax %%<keyword>: parameter values.

EPSF Encapsulated PostScript File format. Looking more closely at the .eps, which resulted from .mp, I found that header comments of a PostScript program starting with

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: llx lly urx ury
%%BeginSetup
%%Endsetup
```

yields the image cropped to the supplied BoundingBox: llx lly urx ury, centred on the page, when processed by Acrobat Pro 7.1.[5] Explicit translation of the origin via ... translate is no longer necessary in an EPSF with a BB around the origin. Very Handy! Not only very handy ... also with better results than my old way via selecting, copying and the reuse of the copy from the clipboard. The dimensions of the BoundingBox can be requested for in the program, to create a perfect cropped illustration.

From Adobe Technical Note #5001 PostScript Language Documents Structuring Convention Specification, the following about the keyword EPSF-3.0

' ... EPSF-3.0 states that the file is an Encapsulated PostScript Format, which is primarily a PostScript language file that produces an illustration. The EPS format is designed to facilitate including these illustrations in other documents. ... '

Appendix H of the LRM version 2 details on EPSF, which by the way is not included in the LRM 3.

PostScript IDE. On the WWW I found the nonfree PSAlter's workbench, a one-window PostScript IDE. In the PostScript FAQs it is mentioned that they provide visual debugging.

I use an editor for creating the source and transform .eps into .pdf by Acrobat Pro 7.1, with 2 windows open:

- the editor
- the map where the PostScript file resides (right mouse click the .eps file yields a pop-up menu with option convert to PDF, which converts the .eps into .pdf and opens the resulting .pdf in my Acrobat Pro 7.1; the %stdout is written to the messsages.log file in the map of Distiller.)

For me Acrobat is an excellent present day interpreter for daily use on my PC. In the mid 90s I used the (black and white) Apple Laser writer.
Those who favour Ghostscript might find the overview http://www.ghostscript.com/doc/7.07/Readme.htm, interesting. GSview does not allow the use of the run operator, apparently, which I use to include my library file.

I tried in TEXworks the preferences option and added Acrobat Pro. Opening in the edit window .eps and processing it by pushing the newly created Acrobat button did open the Acrobat window, yes, but ... empty, without the results. What did I do wrong?

A standing wish: Jonathan Kew, or somebody else, do come up with a nice PostScript...MP IDE similar to the TEXworks IDE, with its handy jumping from a line in the edit window to the corresponding place in the pdf window, and vice versa, its dictionaries (to be added by the user), and ... its promising scripting facilities.

I'm not aware of a MF, respectively MP, pleasing IDE.[6] But ... TEXies can choose for ConTEXt processing in TEXworks, with embedded MetaPost processing on the fly.

The main reasons for me to program in PostScript are:

- I like the powerful graphics facilities
- it is well-suited for my purposes, and ...
- I want to experience once more whether it is more difficult to write in my use of PostScript, which I call minimal PostScript, than in MF or MP, in analogy to minimal markup in TEX versus e.g. the use of LaTEX.[7]

For stringed surfaces emulation PostScript lacks MF's, and MP's inherited, point of operator. No problem, I added the procedure tOnSpline, see later.

A definitely nice feature of MetaFont, respectively MetaPost, I came across while working on this note, is the flexibility of the path specification. In PostScript only B-cubics can be added to the current path, obeying the rigid specification of control points. B-cubics are not provided for by just specifying a set of points lying on the polynomial. But ... with a little bit of Math ... maybe, sometime... someday...

The advantage of working in PostScript directly is that viewing PostScript is a 1-step process, while MP, the TEXworld's PostScript preprocessor, requires 2-steps. Moreover, the resulting .eps from MP is less intelligible and verbose. For example LINEAR CONSTRUCTION IN SPACE No 2 in MP, took 120 lines with the resulting PostScript 800+ lines (mainly caused by unwinded loops). I expect the hand-written PostScript to take an odd 100 lines, roughly of equal size and just as intelligible as the MP version, see Appendix 4.

PostScript output from Adobe Illustrator, which I have used for converting .jpg into .eps, is really unintelligible: lots of preceding obscure definitions, but ... one can use the 'meat' in it as encapsulated PostScript. I'll come back on the matter another time. For the impatient, see Acumen J. Nov 2004, or better still just use Photoshop, or ... for the conversion.

The super reduction in processing steps, is provided for in Hans Hagen's ConTEXt, where MP pictures are processed on the fly, while TEXing. The only thing I miss

is that he does not explain how he accomplished his tricks, for example shading and transparency. At the BachoTEX2010 Taco Hoekwater showed the inclusion of PostScript in LuaTEX, exciting. Note that inclusion of `.eps` in pdfTEX is not allowed.

My minimal use of EPSF. In the program, given in Appendix 2, I made use of the PostScript operators
`closepath curveto def eofill exch fill lineto moveto run setlinewidth setdash showpage translate`
next to the self-written procedures
`s ptp tOnSpline`.
Not so much isn't it?

Choice of curves

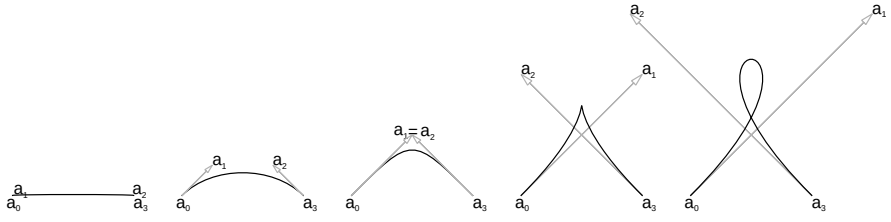
For each hypotenuse, for each of the inner sides of the triangles and for each of the legs of the cap and cup, I use just one B-cubic. The (control) points which characterize the spline have to be chosen, judiciously.

Splines

In PostScript a spline, a Bézier cubic, is characterized by the begin point a_0, the control points a_1 and a_2, also called handles, and the end point a_3. In Java one can drag the handles and watch the effects, interactively. Nice.

> *Splines are the important 20th century's Math time-dependent functions, comparable to the 19th century's Fourier series Math for approximations.*

The control point `a1` lies on the tangent to the spline in a0, and the control point `a2` lies on the tangent to the spline in a3. The points `a0` and `a3` and the angles of the tangents to the spline at these points are not enough to describe the spline uniquely: the size of the handles also matters, as explained in Manning(1972).[8] The control points stand for the angle of the tangents and the size of the handles and therefore determine uniquely the spline.



With `a0` as `currentpoint` a B-cubic, characterized by `a0, a1, a2, a3`, is appended to PostScript's (internal) `path` by `a1 a2 a3 curveto`.

Mathematical formula of a spline. Splines as used in PostScript, and in MP, are Bézier cubics. (Adobe in the Red Book of 1985 mentions along with the operator `curveto` that Bézier cubics are added to the `currentpath`).

These 3rd degree polynomials are a linear combination of 3rd degree Bernstein basis polynomials, which were discovered in 1912 by Bernstein. A n^{th} degree Bernstein basis polynomial reads $B_{\nu n}(t) = \binom{n}{\nu} t^\nu (1-t)^{n-\nu}$, $\nu = 0,1,...,n$.

A Bézier cubic — a linear combination of 3rd degree Bernstein basis polynomials — with begin point a_0, control points a_1, a_2, and end point a_3 reads

$$\begin{aligned} z(t) =& (1-t)^3 a_0 + 3(1-t)^2 t\, a_1 + 3(1-t) t^2\, a_2 + t^3 a_3 \\ & \text{with} \quad z, a_0, a_1, a_2, a_3 \in \mathbb{R}^2,\ t \in [0,1] \end{aligned} \tag{1}$$

The common representation for evaluation, apart from the Horner scheme, reads

$$z(t) = A\,t^3 + B\,t^2 + C\,t + D \quad \text{with} \quad z,A,B,C,D \in \mathbb{R}^2,\ t \in [0,1] \tag{2}$$

The coefficients A,B,C,D are linear in a_0,a_1,a_2,a_3

$$\begin{aligned} A &= \ a_3 - 3a_2 + 3a_1 - a_0 \\ B &= 3a_2 - 6a_1 + 3a_0 \\ C &= 3a_1 - 3a_0 \\ D &= \ a_0. \end{aligned}$$

The above is implemented in `tOnSplineclassic`, which yields the point of the spline, characterized by `a0, a1, a2, a3`, for the time variable `t` by the Horner scheme.[9] PostScript only paints the paths or regions to the current page, it does not provide an operator for evaluating the B-cubic.

De Casteljau's algorithm. Bogusław Jackowski drew my attention to this algorithm for evaluating B-cubics, which is (more generally) discussed on `http://en.wikipedia.org/wiki/De_Casteljau's_algorithm`.
Formula (1) can be written in the de Casteljau representation

$$\begin{aligned} z(t) = (1-t)\Big((1-t)\Big((1-t)a_0 + t\,a_1\Big) + t\,\Big((1-t)a_1 + t\,a_2\Big)\Big) + \\ +t\Big((1-t)\Big((1-t)a_1 + t\,a_2\Big) + t\,\Big((1-t)a_2 + t\,a_3\Big)\Big) \end{aligned}$$

The (vector) value $z(t) = a_{0123}$, of the B-cubic characterized by a_0,a_1,a_2,a_3 for the value of the (time) variable t can algorithmically also be described as

$$\begin{matrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{matrix} \rightarrow \begin{matrix} a_{01} = a_0\,(1-t) + a_1\,t \\ a_{12} = a_1\,(1-t) + a_2\,t \\ a_{23} = a_2\,(1-t) + a_3\,t \end{matrix} \rightarrow$$

$$\rightarrow \begin{matrix} a_{012} = a_{01}\,(1-t) + a_{12}\,t \\ a_{123} = a_{12}\,(1-t) + a_{23}\,t \end{matrix} \rightarrow a_{0123} = a_{012}\,(1-t) + a_{123}\,t$$

Implementation of de Casteljau's algorithm by Bogusław Jackowski and Piotr Strzelczyk for evaluating a spline at point t.

```
/mediation {% a b t ==> c
            % c = a * (1-t) + b * t, a weighted average of a and b
dup 1 exch sub 4 -1 roll mul
3 1 roll mul add
} bind def
```

```
/tOnSpline{{% Purpose: t on spline a0,a1,a2,a3 ==> x y
% implementation of the De Casteljau's algorithm
%
% t: value in [0,1]
% a0 a1 a2 a3: point pairs which characterize the spline
%
% in MF lingo: given pairs a0, a1, a2, a3, and a real number t, 0<=t<=1;
%              we want to compute
%    t[ t[ t[a0,a1],t[a1,a2]],t[t[a1,a2],t[a2,a3]]]
% ==>
```

```
% x(t) y(t)
tOnSplinedict % look up the name and push the dictionary on the operand stack
begin % and move tOnSplinedict dictionary from the operand stack to the d-stack
/a3y exch def /a3x exch def
/a2y exch def /a2x exch def
/a1y exch def /a1x exch def
/a0y exch def /a0x exch def
/t exch def
/a01x a0x a1x t mediation def    /a01y a0y a1y t mediation def
/a12x a1x a2x t mediation def    /a12y a1y a2y t mediation def
/a23x a2x a3x t mediation def    /a23y a2y a3y t mediation def
/a012x a01x a12x t mediation def /a012y a01y a12y t mediation def
/a123x a12x a23x t mediation def /a123y a12y a23y t mediation def
   a012x a123x t mediation       a012y a123y t mediation
end% pop the tOnSplinedict dictionary off the d-stack
}def
/tOnSplinedict 20 dict def%create dictionary with name tOnSplinedict
```

Note. The created variables are 'local', i.e. only made available in the dictionary `tOnSplinedict`, which is popped off the d-stack on leaving the procedure. All important for library procedures.

In MP splines can also be characterized by the begin point, end point, and the tangents at these points (with a wired-in assumption of the smoothness). Even the concept tension, inherited from MF, is implemented such that wired-in smoothness can be overruled.

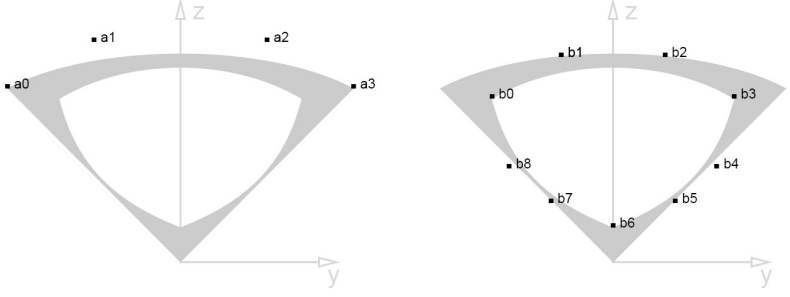
Classically a 3rd degree scalar polynomial

$$P_3(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

is specified by 4 points of the polynomial, or by 2 points and 2 derivative values. The coefficients of the polynomial, a_0,a_1,a_2,a_3, are obtained by solving a 4 x 4 system of linear equations. The latter can be done in MetaPost (and MetaFont) on the fly, albeit without paying attention to numerical stability, i.e. without pivoting. I have not written a numerical stable 4 x 4 linear equation solver in PostScript ... yet; 3 x 3, and 2 x 2, yes; they are included in my PSlib library.

Choice of the data

Choose as origin the centre of the foot. Locate the lower triangle in the yz plane. For the height of the object I decided on 60 pts (nearly an inch), meaning $top = (0,0,60)$, which entails that the height of the triangle is 30 pts. Below at left the points which characterize the outer hypotenuse, at right the points which characterize the inner curves.



For the length of the legs of the triangles I took $25\sqrt{2}$ pts, meaning $a_0 = (0,-25,25)$, and $a_3 = (0,25,25)$, i.e. $-a_y = a_{3y} = 25$.

For the y coordinates of the control points a_1 and a_2, I chose the values -12.5 pts, respectively 12.5 pts. The z coordinates of a_1 and a_2 follow from the equation (2) and that the hypotenuse should touch (0,0,30)

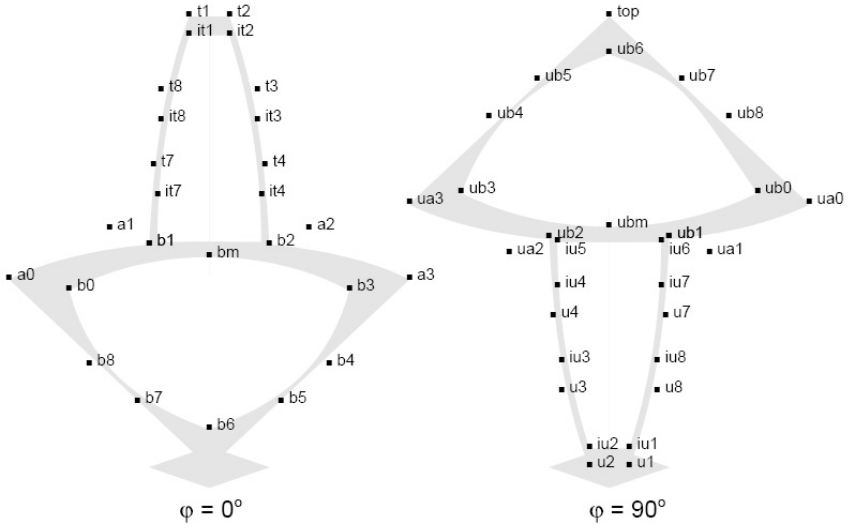
$$z(.5) = \frac{A_z}{8} + \frac{B_z}{4} + \frac{C_z}{2} + D_z \quad \text{with} \quad z(.5) = 30$$

which yields

$$a_{1z} = a_{2z} = (4z(.5) - a_z)/3 = 31\tfrac{2}{3} \rightarrow a_1 = (0, -12.5, 31\tfrac{2}{3}), a_2 = (0, 12.5, 31\tfrac{2}{3}).$$

The data points for the inner triangle, $b_0, \ldots b_8$, must be chosen such that the curves are pleasing and symmetric around the axis of the object; not critical. The data points for the upper triangle are rotated and mirrored values of those of the lower triangle.

I chose for cap and cup data point values which yield curves which looked pleasing to me. Visually, the data points are given in the figure below (for the values see the program in Appendix 2)



By the way, the dots and names above are printed by the following, interesting, mean and lean PostScript code, where the names of the projected points are supplied as strings in an array on the stack. The strings are used as such by `show` and converted to the name of the data points, which they represent.

```
/dotsandnames{%[ str,..., str]==>
%in the str a name, which stands for a pair, such as in pair moveto
%centershow is my centred show, H10pt stands for Helvetica 10 pts
%already scaled
{dup cvn load exec moveto (.) H10pt setfont centershow
                              H12pt setfont show}forall
}def
%invoke
[(a0) (a1) (a2) (a3) (top)... ] dotsandnames
```

Explanation line by line: In a `forall` loop each string is duplicated (remember that each string in the array is put on the stack in turn, in the order as given in the array, which is different from other languages, where parallel execution of the elements of the `forall` loop is allowed, be aware!) after which one copy is converted into a name, loaded and executed to yield the coordinates of the data point for the positioning, and the dot is (centered) painted by `centershow` to the current page; next in the loop the duplicated string is painted by `show` to the current page. Neat!

But ... sometimes the names can better be shown left, as in the frame of LINEAR CONSTRUCTION IN SPACE No 1, see Appendix 3.

Implementation I chose for each projected point a definition, which for `top`, for example, reads

```
/top { 0 0 60 s ptp} def% s=scaling, analogue to the use of 60 inch
```

The advantage is mean and lean correct code with deferred execution. The disadvantage is that each point is evaluated each and every time when needed. For my toy problems, on nowadays PCs, this is not relevant; clear, intelligible, and correct code is more important than execution speed.

A more efficient, but verbose, version reads

```
0 0 60 s ptp /topy exch def /topx exch def   /top { topx topy } def
```

The points named by `a0, a1, a2,...` in the program denote the projected spacial data a_0,a_1,a_2,... . The (projected) triangles are called `lowertorsiontriangle` and `uppertorsiontriangle` and implemented as follows

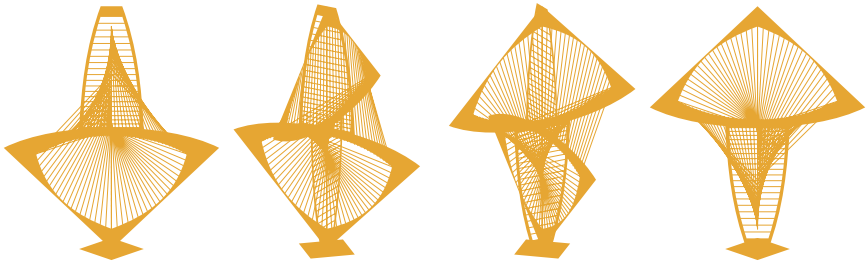
```
/lowertorsiontriangle{origin moveto  %outer path
       a0 lineto
       a1 a2 a3 curveto closepath
       b6  moveto                    %inner path
       b7 b8 b0 curveto
       b1 b2 b3 curveto
       b4 b5 b6 curveto closepath
} def
```

Stringing

Connecting points of the splines by straight lines yields the stringing, the stringed surfaces. The points on the spline are obtained by invokes of `tOnSpline`, see the procedure `dostringing` in the program given in Appendix 2.

Results

Below the emulations seen under azimuths 0°, 30°, 60°, 90°, and inclination 20°



Variations

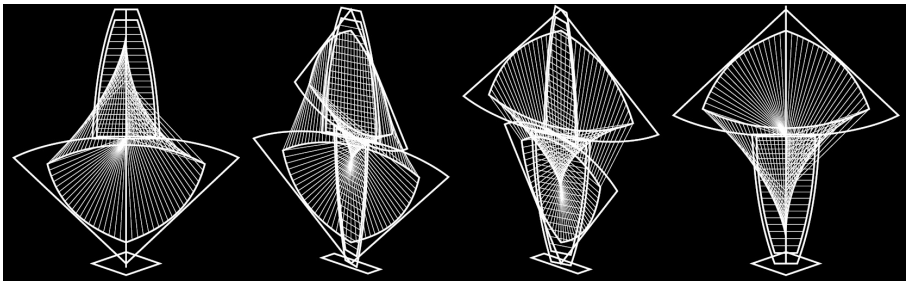
Gabo himself constructed variations, where the sides of the triangles are connected to the outer sides of cap, respectively cup. This can be coded easily, but I expect it to blur the picture by too many lines.

On closer inspection I noticed that in the centre Gabo had a horizontal metal square width sides of 10 cm or so. The hypotenuses were split in the middle and connected to the corners of the square.

Other emulations may use circular arcs as inner sides of the triangles (actually this was the case in the real thing).

On the other hand, one may vary by using curves instead of lines for the sides of the triangles.

Below the results of the outline version in reverse video, a fibre figure, which I consider beyond Gabo, because in the emulations I don't have to worry about construction necessities, although they much resemble his LINEAR CONSTRUCTIONs in perspex.



Inherent are his variations in the construction process from models in card or plastic to real objects made from wood, metal, perspex, plastic, ... of varying size.

Gabo's ultimate variation on the Torsion theme is his fountain in front of the Thomas hospital in London, opposite the Big Ben on the other bank of the river Thames.

In this kinetic art fountain the 'stringed' surfaces change, because the strings are water jets squirted under slowly varying pressure. The revolting time is 10 minutes.

In the beginning of this century I got an idea how to emulate this fountain (material: white curved plastic electricity tubes and a rotating lawn sprinkler) for my quarter circle garden pond.

I guess that the construction principle differs: my lawn sprinkler revolves on a water bed and due to the action=reaction principle.

Conclusions

Though PostScript lacks an explicit path data structure the TORSION object was easily emulated. The points of a Bézier cubic are calculated by the de Casteljau algorithm implemented by Bogusław Jackowski and Piotr Strzelczyk. Such an operator is not provided for in PostScript.

The use of ready-made EPSF procedures is not difficult at all, as can be witnessed from the included programs; to write robust EPSF procedures is a different story. A library of correct, intelligible EPSF procedures is what we need.

Bézier cubics are invariant under (oblique parallel) projection, which is according to Bogusław Jackowski obvious.

To program illustrations in PostScript is no more difficult than to program in MP, or MF, but ... one has to understand the concepts, thoroughly. For me the use of PostScript is easier than the use of MP.

Starting an EPSF program with

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox ...
%%Beginsetup
%%Endsetup
```

yields `.pdf` pictures, cropped to the specified BoundingBox, centred.

The handling of hidden lines is circumvented.

The use of `pdf optimizer` of Acrobat Pro 7.1 resulted in a ≈ 30 times smaller `.pdf` file of this note.[10]

In TEXworks' editor window I use the monotype Lucida console font, where all characters have the same width (it is default in MS Wordpad) in order to keep verbatim texts vertically aligned, but it did not work as expected; 16 pts is a convenient magnification for me. Bookman Old Style (gave good results with respect to alignment in verbatims) and Baskervile Old Face are also monotype and pleasant for the eyes. The pdf viewer used by TEXworks is not so good as Acrobat: sometimes annotations in my illustrations don't show up.

Never throw data away, even after 15 years they may still be of value; beware when you replace your computer.

TEXing this document required little, minimal plain TEX markup. Interesting, although not suitable for full-automatic typesetting with its shrinking and stretching, its automatic splitting into pages, ... is the inclusion of a picture next to the verbatim listing. (A practice borrowed from the Blue Book. I did not use Phil Taylor's `\parshape` preprocessor for flowing text around illustrations, presented at BachoTEX 2009, because I was only after placing illustrations in the open space at the right of (narrow) verbatim listings.) A way how to do this is essentially explained in

the TEXbook on p389, which I implemented in the macros `\insertjpg`, respectively `\insertpdf`. The TEXnique I used earlier in my plain TEX Turtle Graphics macros, which are not suited for graphics in TEX: EPSF is much better and more general for the description of (single) illustrations, to be included in AnyTEX documents. I used the wrong tool for my graphics: TEX as American screwdriver, to paraphrase A. Perlis.

When I think I'm finished it turns invariably out that I'm only halfway.

I don't expect you, kind reader — who does not suffer of the disease of our times: lack of time — to come up with a Gabo of your own (not that difficult just design a frame and do the stringing), but ... you may profit from the projection technique, the examples of EPSF programming, the communicated experiences, and hopefully you have become familiar with some of Gabo's works and last but not least enjoy the illustrations.

Æsthetics and effectiveness of the message, cultural contexts? My message: PostScript can be used gracefully, æsthetically, and effectively to emulate a class of Gabo's objects, and undoubtedly some more. TEXies should catch up.

Wouldn't it be nice to have holographic 3D projections of Gabo's constructions?

Afterthoughts

The projected frame resembles a font character. Would the program, or me in writing the program, have benefited from 'subscripting' more in the spirit of MF's general `suffix`? I might have used a_i, inner a, instead of `b` and subscripted it by numbers: i.e. have used variables `a0i, a1i,...` instead of `b0, b1, ...`. I did do this for cap and cup and followed Knuth's convention in LINEAR CONSTRUCTION IN SPACE No 1 to begin with, see Appendix 3.

A standing wish An IDE for PostScript, and MP, in the spirit of TEXworks I would welcome. Why not distribute it on the TEXlive DVD?

Daydreaming If only I could handle gradients in PostScript gracefully, I might emulate Gabo's CONSTRUCTION HEAD No 3 (Head in a Corner Niche, which I watched in MOMA, NY) although it has a much different character than the stringed surfaces. Gradients I do for the moment by the PostScript level 1 technique. In PS3 really efficient shading gradients are accounted for. Quite substantial, looks complex. Work to do!

Further reading

Instead of a bibliography a selection which I experienced useful:

- Adobe's Red, Green and Blue books, an absolute must.
  `http://www-cdf.fnal.gov/offline/PostScript/`
  From the preface of the Green Book: PostScript Language Program Design is intended to provide a solid background for developing software in the PostScript language — not just getting a program to work, but thoroughly designing it from top to bottom ...
  ... The sample programs contained in each chapter are intended to be directly usable. They are examples, but they are not trivial ones. Each program has been

carefully designed and debugged and should provide an excellent foundation for an industrial-strength printer **driver** ...
Disclaimer: For EPSF usage not so appropriate.

- Adobe PostScript 3 Fonts Set ... come standard with 136 distinctive and stylish fonts, including those packaged with the leading operating systems...
Disclaimer: I did not succeed in finding these fonts in Acrobat Pro 7.1.
Puzzling message to `%stdout`*: Helvetica not found, using Font Substitution. Font cannot be embedded.*
- Digital Acumen Journal: `http://www.acumentraining.com`, by John Teubert, for free. Top!
- Adobe Technical Note #5002, Encapsulated PostScript File Format Specification `http://partners.adobe.com/public/developer/en/ps/5002.EPSF_Spec.pdf`
- Don Lancaster's PostScript use: `http://www.tinaja.com`, for free. Informative.
- PostScript FAQ `http://en.wikipedia.org/wiki/Wikibooks:PostScript_FAQ`

On the other hand, I myself have published in MAPS a few articles on, cq related to, PostScript:

- Just a little bit of PostScript, MAPS 17, contains nice, if I may say so, examples you don't see that often.
- Stars around I & Stars around II, MAPS 18, after Jackowski's MetaFont lectures. Jacko's example of the creation of the 2 character `OK` font is much in the spirit of the creation of an analytic font in PostScript as given in the Blue Book Program 20.
- Tiling in PostScript and MetaFont—Escher's wink MAPS 19.
- TEX education—a neglected approach MAPS 39. Presented at the EuroTEX&ConTEXt 2009 and rehearsed at the BachoTEX 2010.
- Circle Inversions MAPS 40, where I introduced among others my `PSlib`.
- à la Mondriaan MAPS 41. Presented at BachoTEX 2010.
- Programming Pearls 2010: π-decimals along a spiral and `The mouse's tail and Alice's tale'.
- Programming Pearls 2011: Pythagoras Tree.

For the Math:

- Lauwerier, H.A.(1987):[11]Meetkunde met de Microcomputer. Epsilon 8. (Projection, hidden lines, projection of crystals, inside-outside, sphere with meridians, toroid ... . However ... no projection of B-cubics. How to project circles and ellipses efficiently was developed while working on this note.)
- Manning J.R.(1972): Continuity conditions for spline curves. Computer Journal, 17,2, p181-186.

For those who want to pass by (La)TEX and do it all in PostScript might find the following interesting
`http://www.cappella.demon.co.uk/bookpdfs/pracpost.pdf`
NTG provides a printing on demand service `http://www.boekplan.nl` for among others copies of MAPS articles.

Acknowledgements

Thank you Naum Gabo for your inspiring art, Hans Lauwerier for your lectures on projection and some more, Adobe for your good old PostScript and `Acrobat` to view it, Don Knuth for your stable plain TEX, Jonathan Kew for the TEXworks IDE, Hàn Thế Thành for your pdfTEX, Bogusław Jackowski for your thorough review, your communicating de Casteljau's algorithm and sending me a copy of Manning's paper,

Piotr Strzelczyk for the implementation of de Casteljau's algorithm, Jos Winnink for comments on an early version of this paper, Henk Jansen for stressing the point to coddle data, which might still be of use after ≈15+ years, Wim Wilhelm for prompting the formula for the curve on the tennis ball, MAPS editors, especially Frans Goddijn, for improving my use of English, and Taco Hoekwater for suggestions and procrusting this note into MAPS format.

Notes

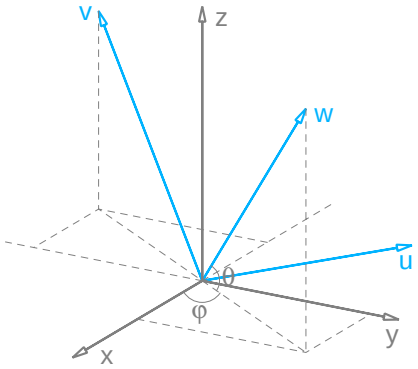
1. Naum Gabo, 1890–1977. Born Naum Borisovich Pevsner. Bryansk. Russian Constructivist. An excellent book about him and his works: Naum Gabo 60 years of Constructivism. Prestel-Verlag 1985, which appeared on the occasion of the retrospective exhibition with the same name at the Dallas Museum of Art, the Art Gallery of Ontario, the Guggenheim Museum NY, the Akademie der Künste Berlin, the Kunstsammlung Nordrhein-Westfalen, the Tate Gallery London. Wikipedia contains a short biography.
2. Courtesy: Courant, R (1936) Differential and Integral calculus II, p175. The astroid was already discussed by J. Bernouilli in 1691.
3. Note that there is no automatic type conversion. For example the index of an array `a` must be of type integer eventually converted by `cvi: a <real expression> cvi <value> put`. This also holds for the integer operands of `idiv`.
4. The procedure texts and the examples are available on the WWW as a UNIX shell archive. I have assembled the procedures into a `BlueBook.eps` library, which is system independent.
5. An undocumented feature? Non-universal?
6. On my PowerMac and Mac Classic of ≈ 1992 (still alive, mind you!), I could view MF pictures on the screen in Blue Sky's MF. I don't have that functionality on my PC. No longer relevant because the resulting bitmaps are outdated.
7. Obeying the famous 80-20% adage: 20%, or less, of the needed energy with 80%, or more, of the results. It is true that LaTeX comes with a wealth of packages. John Hobby has donated the powerful `boxes` for flowcharts, and `graph` for drawing graphs in MP.
8. This is different from a scalar (3^{rd} degree) polynomial $P_3(x)$, where 2 values and 2 derivatives determine the polynomial uniquely.
9. Well-known time representations of curves are the Lissajous figures defined by (A sin(at+δ), B cos(bt)).
10. Courtesy Péter Szabó, EuroTeX 2009.
11. For Lauwerier's biography see the general site about Dutch mathematicians: `http://bwnw.cwi-incubator.nl/cgi-bin/uncgi/alf`

My case rests, have fun and all the best.

Kees van der Laan
Hunzeweg 57, 9893PB Garnwerd, Gr, NL
email: kisa1@xs4all.nl

Appendix 0 Projection formula

Let us choose a Cartesian orthonormal (projection) coordinate system uvw, with the w axis in the direction (ϕ,θ), the v axis in the wz plane orthogonal to the w axis. Because the v and w axes are in the plane wz the u axis is in the xy plane, with direction according to the right-screw rule.



uvw coordinate system
with $\mathbf{e}_u, \mathbf{e}_v, \mathbf{e}_w$ unit vectors
$\mathbf{e}_w$ in the direction (ϕ,θ) (Puvw)
$\mathbf{e}_v$ in the zw plane, $\mathbf{e}_v \perp \mathbf{e}_w$
$\mathbf{e}_u$ in the xy plane, $\mathbf{e}_u \times \mathbf{e}_v = \mathbf{e}_w$

A point **P** can be described in the coordinate systems as

$$\mathbf{P} = x\mathbf{e}_x + y\mathbf{e}_y + z\mathbf{e}_z = u\mathbf{e}_u + v\mathbf{e}_v + w\mathbf{e}_w$$

If the unit vectors $\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z$ are projected in the uvw coordinate system we obtain

$$\begin{aligned}\mathbf{P} = {} & x(a_{11}\mathbf{e}_u + a_{21}\mathbf{e}_v + a_{31}\mathbf{e}_w) + \\ & y(a_{12}\mathbf{e}_u + a_{22}\mathbf{e}_v + a_{32}\mathbf{e}_w) + \\ & z(a_{13}\mathbf{e}_u + a_{23}\mathbf{e}_v + a_{33}\mathbf{e}_w) \\ = {} & u\mathbf{e}_u + v\mathbf{e}_v + w\mathbf{e}_w\end{aligned}$$

Rearranging terms and equating coefficients yields the change of coordinates: $(x,y,z) \rightarrow (u,v,w)$, which in matrix notation reads

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Deleting the w component yields the projection coordinates in the uv plane, the projection plane. Substituting the values for a_{ij} in terms of ϕ and θ yields the projection

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\sin\phi & \cos\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Properties

- because of the orthonormal coordinate systems the coefficients of the matrix A obey the relations

$$\sum_{k=1}^{3} a_{ik}a_{jk} = \delta_{ij} \begin{cases} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{cases}$$

□ because a similar reasoning can be applied to transforming the uvw system into the xyz system, the inverse matrix equals the transposed matrix, i.e. $A^{-1}=A^{t}$.

Factorization of the projection matrix

$$\begin{pmatrix}1&0&0\\0&1&0\end{pmatrix}\begin{pmatrix}1&0&0\\0&\sin\theta&\cos\theta\\0&-\cos\theta&\sin\theta\end{pmatrix}\begin{pmatrix}-\sin\phi&\cos\phi&0\\-\cos\phi&-\sin\phi&0\\0&0&1\end{pmatrix}=\begin{pmatrix}-\sin\phi&\cos\phi&0\\-\cos\phi\sin\theta&-\sin\phi\sin\theta&\cos\theta\end{pmatrix}$$

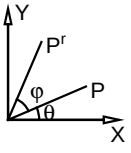
The left factor is rotation over $\theta-\frac{\pi}{2}$; the other factor

$$\begin{pmatrix}-\sin\phi&\cos\phi&0\\-\cos\phi&-\sin\phi&0\\0&0&1\end{pmatrix}=-\begin{pmatrix}\cos(\frac{\pi}{2}-\phi)&-\sin(\frac{\pi}{2}-\phi)&0\\\sin(\frac{\pi}{2}-\phi)&\cos(\frac{\pi}{2}-\phi)&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&0&-1\end{pmatrix}$$

is a composition of inversion of the z coordinate and rotation over $\frac{\pi}{2}-\phi$.

Background: rotation over ϕ in xy plane The coordinates (x',y') of a point $P_{r\,\theta}(x,y)=(r\cos\theta,r\sin\theta)$ rotated over ϕ read

$$\begin{pmatrix}x'\\y'\end{pmatrix}=\begin{pmatrix}r\cos(\phi+\theta)\\r\sin(\phi+\theta)\end{pmatrix}=\begin{pmatrix}r(\cos\phi\cos\theta-\sin\phi\sin\theta)\\r(\cos\phi\sin\theta+\sin\phi\cos\theta)\end{pmatrix}=\begin{pmatrix}\cos\phi&-\sin\phi\\\sin\phi&\cos\phi\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}$$



Lauwerier used the projection matrix $\begin{pmatrix}-39&52&0\\-20&-15&60\end{pmatrix}$ because he was after a pleasant projection, and his μ-computer was not powerful enough to facilitate animation, even in BASIC.

Appendix 1 Projection of curves

Invariance of Bézier cubics under projection

Observation A projection of a Bézier cubic can be obtained by projecting the begin point, the control points, and the end point, and use these to build the (projected) spline.

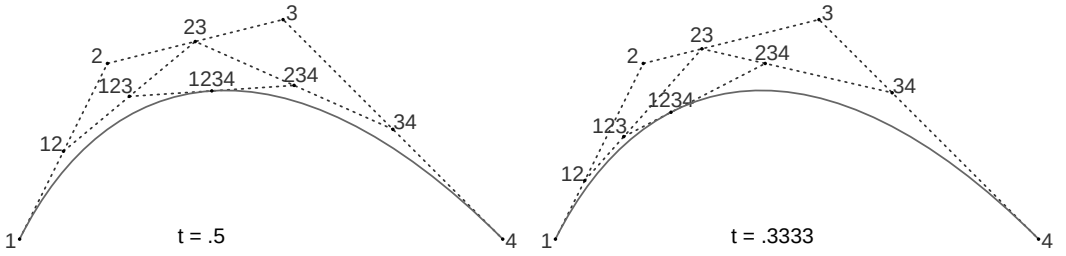
Proof A Bézier cubic is characterized by four points z_1,z_2,z_3,z_4, the begin point, the control points and the end point. A point on the curve, z_{1234} the third-order midpoint, is obtained by

$$\begin{matrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{matrix} \rightarrow \begin{matrix} z_{12} = \frac{1}{2}[z_1, z_2] \\ z_{23} = \frac{1}{2}[z_2, z_3] \\ z_{34} = \frac{1}{2}[z_3, z_4] \end{matrix} \rightarrow \begin{matrix} z_{123} = \frac{1}{2}[z_{12}, z_{23}] \\ z_{234} = \frac{1}{2}[z_{23}, z_{34}] \end{matrix} \rightarrow z_{1234} = \frac{1}{2}[z_{123}, z_{234}]$$

where $\frac{1}{2}[z_1,z_2]$ means the midpoint of the line through z_1 and z_2.

To get the remaining points of the curve, for eample for the time variable $t = \frac{1}{3}$, determined by z_1,z_2,z_3,z_4

repeat the same construction on z_1,z_{12},z_{123},z_{1234} and z_{1234},z_{234},z_{34},z_4, ad infinitum (Courtesy The MF book p13).



If we project the four initial points and construct the polynomial in the projection plane from the projected points, the same curve will result as if we had projected each point of the original polynomial, because the construction lines and their midpoints are invariant. qed.

Note the above shows that Knuth was already aware of what became known as the de Casteljau algorithm for evaluation of B-cubics.

Projection of circles and ellipses
A point on a circle obeys

$$(x,y) = (r\cos t, r\sin t) \quad t \in [0,2\pi] \quad \text{with} \quad x^2 + y^2 = r^2 \quad r \text{ the radius and } (0,0) \text{ the centre.}$$

The above circle is in PostScript constructed as path by `0 0 r 0 360 arc`.
A point on an ellipse obeys

$$(x,y) = (a\cos t, b\sin t) \quad t \in [0,2\pi] \quad \text{with} \quad \frac{x^2}{a} + \frac{y^2}{b} = 1 \quad \text{with } a \text{ and } b \text{ half the axes.}$$

From this we can understand that the an elliptical path can be obtained in PostScript by
`a b scale 0 0 1 0 360 arc` (Courtesy the Blue Book p55, MetaFontbook p123).

The projection (u,v) of a point (x, y, z) on a circle in 3D obeys

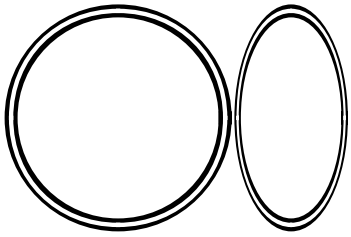
$$(u,v) = (-x\sin\phi + y\cos\phi,\ x\cos\phi\sin\theta - y\sin\phi\sin\theta + z\cos\theta)$$

which does not look like a composition of scaling and/or rotation, but actually it is, as communicated by Bogusław Jackowski

Projection of a circle or ellipse can be done by sampling and connecting the sample points in the projection plane by `linetos`, of course. But ... what if we project B-cubic approximations? (I did not find time yet to go for the approach suggested by Bogusław Jackowski: find the corresponding transformation matrix.)

Projection of approximated circles and ellipses Sampling of a circle for projection requires 360 samples, say, dependent on the wanted accuracy. What if we approximate the circle by 4 suitably chosen B-cubics?

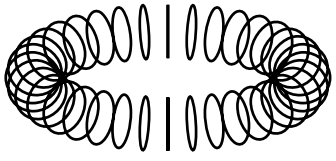
```
...
% .5 1 scale
0 0 100 0 360 arc 12 setlinewidth stroke %circle
4{-100 0 moveto -100 57 -57 100 0 100 curveto
  90 rotate}repeat% 4 B-cubics approximation of circle
1 setgray 4 setlinewidth stroke
...
```



The white approximation of the black circle, respectively ellipse, is good enough, for my projection purposes.

Why Handles of size 57? This value results from the requirement that the circle with radius 1 should pass through $\frac{1}{\sqrt{2}}(1,1)$, see below. It demonstrates that a user should have some knowledge of B-cubics.

Conclusion The projection of a circle, respectively ellipse, can be efficiently approximated by projection of 4 suitably chosen B-cubics. Instead of 360 sample points, say, only 12 points have to be projected. Not that relevant for my toy problems — I did not notice the difference in my LINEAR CONSTRUCTION IN SPACE No 1 & 2 — and nowadays PCs, but ... it is a factor 30, or so, more efficient, and no more cumbersome than the full sampling approach.



A nice example of repeatedly drawing projected circles is the toroid impression

A professional starts where an amateur ends In the MetaFont book p263 Knuth approximates the quarter circle by

```
quartercircle=(right{up}..(right+up)/sqrt2..up{left}) scaled .5
```

which is better, because he attacks the point of the greatest deviation by requiring that the spline should pass through $\frac{1}{\sqrt{2}}(1,1)$, unscaled. For the full circle (with unit diameter) he splices rotated copies of the quarter circle.

In PostScript we don't have such a path construction operator.

The requirement that the B-cubic, see Bézier formula (1), specified by $(1,0),(1,\delta),(\delta,1),(0,1)$ should pass for $t = .5$ through $\frac{1}{\sqrt{2}}(1,1)$ yields $\delta \approx .573$.

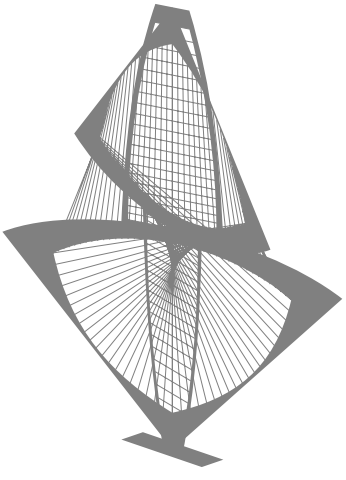
Appendix 2 TORSION emulation in EPSF

Conceptually the program, the script in PostScript lingo, is a handful of lines.

```
/theta 20 def  /phi 45 def  /scalingfactor 4 def
.5 setgray 0.5 setlinewidth %settings
foot fill
lowertorsiontriangle eofill cap eofill
uppertorsiontriangle eofill cup eofill
stringing stroke% paint the stringed surfaces to the current page
0 setgray annotations
showpage %print the current page
```

preceded by `%!PS-Adobe-3.0 EPSF-3.0`, DSC Comments, the Prolog (with the inclusion of the library, the ad hoc procedures cq data), and eventually followed by the Trailer part and closed by `%%EOF`.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Emulation of Naum Gabo Torsion
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%CreationDate: Februari 2011
%%BoundingBox: -35 -10 35 65
%%BeginSetup
%%EndSetup
(C:\\PSlib\\PSlib.eps) run %used from lib: ptp ,  tOnSpline, s (scaling) and default scalingfactor 1
%Specification of data in 3D and projections in 2D, dependent on phi, theta and scaling
%lowertorsiontriangle data
/a0 { 0 -25   s  25     s  ptp} def %outer points
/a1 { 0 -12.5 s  31.67  s  ptp} def
/a2 { 0  12.5 s  31.67  s  ptp} def
/a3 { 0  25   s  25     s  ptp} def
/origin { 0 0 } def
/b0 { 0 -17.5 s  23.5   s  ptp} def%inner points
/b1 { 0  -7.5 s  29.5   s  ptp} def
/bm { 0   0      28     s  ptp} def
/b2 { 0   7.5 s  29.5   s  ptp} def
/b3 { 0  17.5 s  23.5   s  ptp} def
/b4 { 0  15   s  13.5   s  ptp} def
/b5 { 0   9   s   8.5   s  ptp} def
/b6 { 0   0       5     s  ptp} def
/b7 { 0  -9   s   8.5   s  ptp} def
/b8 { 0 -15   s  13.5   s  ptp} def
%uppertorsiontriangle, 90 rotated, upside down data
/ua0 { -25   s 0  35    s  ptp} def%outer points
/ua1 { -12.5 s 0  28.34 s  ptp} def
/ua2 {  12.5 s 0  28.34 s  ptp} def
/ua3 {  25   s 0  35    s  ptp} def
/top { 0 0 60 s  ptp} def
/ub0 { -18.5 s 0  36.5 s   ptp} def%inner points of uframe
/ub1 {  -7.5 s 0  30.5 s   ptp} def
/ub2 {   7.5 s 0  30.5 s   ptp} def
/ub3 {  18.5 s 0  36.5 s   ptp} def
/ub4 {  15   s 0  46.5 s   ptp} def
/ub5 {   9   s 0  51.5 s   ptp} def
/ub6 {   0     0  55   s   ptp} def
/ub7 {  -9   s 0  51.5 s   ptp} def
/ub8 { -15   s 0  46.5 s   ptp} def
/ubm {   0     0  32   s   ptp} def
%cap outer
/t1 { 0 -2.5 s 60 s  ptp} def
/t2 { 0  2.5 s 60 s  ptp} def
/t3 { 0  6   s 50 s  ptp} def
/t4 { 0  7   s 40 s  ptp} def
```



φ = 30° θ = 20°

```
/t5 { b2 } def
/t6 { b1 } def
/t7 { 0 -7   s 40 s  ptp} def
/t8 { 0 -6   s 50 s  ptp} def
%cap inner
/it1 { 0 -2.5 s 57.5 s  ptp} def
/it2 { 0  2.5 s 57.5 s  ptp} def
/it3 { 0  6   s 46   s  ptp} def
/it4 { 0  6.5 s 36   s  ptp} def
/it5 { 0  6.5 s 30   s  ptp} def
/it6 { 0 -6.5 s 30   s  ptp} def
/it7 { 0 -6.5 s 36   s  ptp} def
/it8 { 0 -6   s 46   s  ptp} def
%cup outer; mirrored vertically around z=30 and rotated
/u1 { -2.5 s 0   0    ptp} def
/u2 {  2.5 s 0   0    ptp} def
/u3 {  6   s 0  10 s  ptp} def
/u4 {  7   s 0  20 s  ptp} def
/u5 { ub2 } def
/u6 { ub1 } def
/u7 { -7 s  0  20 s  ptp} def
/u8 { -6 s  0  10 s  ptp} def
%cup inner
/iu1 { -2.5 s 0  2.5 s ptp} def
/iu2 {  2.5 s 0  2.5 s ptp} def
/iu3 {  6   s 0 14   s ptp} def
/iu4 {  6.5 s 0 24   s ptp} def
/iu5 {  6.5 s 0 30   s ptp} def
/iu6 { -6.5 s 0 30   s ptp} def
/iu7 { -6.5 s 0 24   s ptp} def
/iu8 { -6   s 0 14   s ptp} def
%frame, formally in x- and y-components of the points
/lowertorsiontriangle{origin moveto %outer path
       a0 lineto
       a1 a2 a3 curveto closepath
       b6  moveto                                %inner path
       b7 b8 b0 curveto
       b1 b2 b3 curveto
       b4 b5 b6 curveto closepath} def
/uppertorsiontriangle{top moveto %outer path
        ua0 lineto
        ua1 ua2 ua3 curveto closepath
        ub6 moveto                           %inner path
        ub7 ub8 ub0 curveto
        ub1 ub2 ub3 curveto
        ub4 ub5 ub6 curveto closepath} def
/foot{-7.5 s     0   0  ptp moveto
         0    -7.5 s 0  ptp lineto
       7.5 s     0   0  ptp lineto
         0     7.5 s 0  ptp lineto
      closepath} def
/cap{ t1 moveto t2 lineto %outer
        t3 t4 b2 curveto
        b1 lineto
       t7 t8 t1 curveto closepath
      it1 moveto it2 lineto %inner
      it3 it4 it5 curveto
      it6 lineto
      it7 it8 it1 curveto closepath
}def
/cup{ u1 moveto u2 lineto %outer
      u3 u4 u5 curveto
      u6 lineto
      u7 u8 u1 curveto closepath
     iu1 moveto iu2 lineto %inner
      iu3 iu4 iu5 curveto
      iu6 lineto
```

```
      iu7 iu8 iu1 curveto closepath
}def
%build the paths for stringed surfaces
/dostringing{ 0 0.05 1.01{/t exch def
  t b0  b8  b7  b6  tOnSpline moveto
  t 2 div ub0 ub1 ub2 ub3 tOnSpline lineto
%
  t 2 div b0  b1  b2  b3  tOnSpline moveto
  t ub3 ub4 ub5 ub6 tOnSpline lineto
%
  t b3  b4  b5  b6  tOnSpline moveto
  t 2 div ub3 ub2 ub1 ub0 tOnSpline lineto
%
  t 2 div .5 add b0  b1  b2  b3  tOnSpline moveto
  t ub6 ub7 ub8 ub0 tOnSpline lineto
%cap
  t it1  it8  it7   it6  tOnSpline moveto
  t it2  it3  it4   it5  tOnSpline lineto
%cup
  t iu1  iu8  iu7   iu6  tOnSpline moveto
  t iu2  iu3  iu4   iu5  tOnSpline lineto
}for} bind def %end dostringing
%
/annotations{-32 -25  moveto
 S12pt setfont (j) show ( = ) H12pt setfont show phi   (  ) cvs show %note the use of phi for its value
   gsave 0 5 rmoveto (o) H7pt setfont show grestore
8 0 rmoveto
 S12pt setfont (q) show ( = ) H12pt setfont show theta (  ) cvs show %note the use of theta for the value
         0 5 rmoveto (o) H7pt setfont show} def
%%EndProlog
%
%---Program--- the script
%
/theta 20 def  /phi 45 def  /scalingfactor 4 def
.5 setgray 0.5 setlinewidth %settings
foot fill
lowertorsiontriangle eofill cap eofill
uppertorsiontriangle eofill cup eofill
dostringing stroke% paint the stringed surfaces to the current page
0 setgray annotations
showpage
%%EOF
```

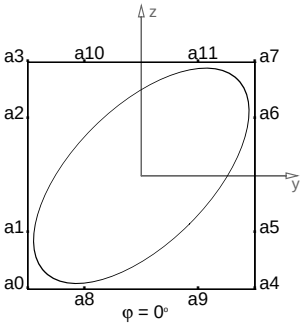
The annotations end all the emulation programs and are not repeated in the other listings, though they differ a little.

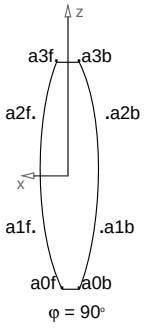
Appendix 3 Linear Construction in Space No 1 emulation in EPSF

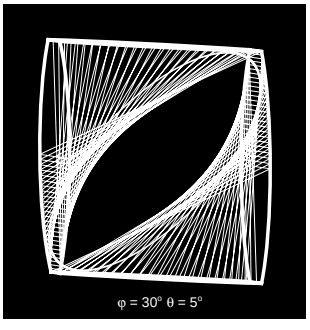
In one afternoon I rewrote the following program as emulation of LINEAR CONSTRUCTION IN SPACE No 1 from scratch in PostScript as EPSF, with subscripting conventions à la MetaFont for the points which determine the frame: the suffix `f` is a mnemonic for in **f**ront of the `yz` plane, the suffix `b` for **b**ehind the `yz` plane. The interior ellipse, which I neglected previously in the MetaFont code, has been accounted for (Gabo needed the interior slab for solidity. The elliptic hole for beauty?).

The program comprises 85 lines (roughly the same as the older `.mp`). The `.eps` result from the 1996 MP source takes 233 lines and is less intelligible.
The listing of the code is in the spirit of the Blue Book: the illustration along with the commented code. For the reader's convenience I have also included the frame with the data points, visually.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Emulation of Naum Gabo Linear Construction in Space No 1
%%BoundingBox: -130 -135 130 135
%%BeginSetup
%%EndSetup
%%Creator: Kees van der Laan
%%CreationDate: Februari 2011
%%DocumentFonts: Helvetica System
%%EndComments
(C:\\PSlib\\PSlib.eps) run
/reversevideo{-130 -135 260 270 rectfill}def%Mimics the BoundingBox
/origin { 0 0 } def
/a0f  { .1 s  -2 s  -2 s ptp  }def%y constant
/a1f  { .6 s  -2 s  -1 s ptp  }def
/a2f  { .6 s  -2 s   1 s ptp  }def
/a3f  { .2 s  -2 s   2 s ptp  }def

/a4f  { .2 s   2 s  -2 s ptp  }def%y constant
/a5f  { .6 s   2 s  -1 s ptp  }def
/a6f  { .6 s   2 s   1 s ptp  }def
/a7f  { .2 s   2 s   2 s ptp  }def

/a8f  { .6 s   -1 s -2 s ptp  }def%z constant
/a9f  { .6 s    1 s -2 s ptp  }def

/a10f { .6 s   -1 s  2 s ptp  }def%z constant
/a11f { .6 s    1 s  2 s ptp  }def

/a0b  { -.2 s  -2 s -2 s ptp  }def%y constant
/a1b  { -.6 s  -2 s -1 s ptp  }def
/a2b  { -.7 s  -2 s  1 s ptp  }def
/a3b  { -.2 s  -2 s  2 s ptp  }def

/a4b  { -.2  s  2 s -2 s ptp  }def%y constant
/a5b  { -.6  s  2 s -1 s ptp  }def
/a6b  { -.6  s  2 s  1 s ptp  }def
/a7b  { -.2  s  2 s  2 s ptp  }def

/a8b  { -.6 s  -1 s -2 s ptp  }def%z constant
/a9b  { -.6 s   1 s -2 s ptp  }def

/a10b { -.6 s  -1 s  2 s ptp  }def%z constant
/a11b { -.6 s   1 s  2 s ptp  }def
%
/frame{
a0f moveto  a1f  a2f  a3f curveto
            a10f a11f a7f curveto
            a6f  a5f  a4f curveto
            a9f  a8f  a0f curveto closepath
```







```
a0b moveto  a1b  a2b  a3b curveto
            a10b a11b a7b curveto
            a6b  a5b  a4b curveto
            a9b  a8b  a0b curveto closepath
a0f moveto a0b lineto
a3f moveto a3b lineto
a4f moveto a4b lineto
a7f moveto a7b lineto
}def
%
/approxellipsestroke{gsave % project approximated ellipse (by 4 B-cubics)
-45 rotate
.6 1.2 scale%kind of ellips
/a0 {0 -2 s    0     ptp} def
/a1 {0 -2 s    1.1 s ptp} def
/a2 {0 -1.1 s  2 s   ptp} def
/a3 {0  0      2 s   ptp} def
/a4 {0  1.1 s  2 s   ptp} def
/a5 {0  2 s    1.1 s ptp} def
/a6 {0  2 s    0     ptp} def
/a7 {0  2 s   -1.1 s ptp} def
/a8 {0  1.1 s -2 s   ptp} def
/a9 {0  0     -2 s   ptp} def
/a10{0 -1.1 s -2 s   ptp} def
/a11{0 -2 s   -1.1 s ptp} def
a0 moveto a1  a2  a3 curveto
          a4  a5  a6 curveto
          a7  a8  a9 curveto
          a10 a11 a0 curveto
          closepath
stroke grestore} def
%
/dostringing{.02 .02 .5001{/t exch def
t       a0f a1f  a2f  a3f tOnSpline moveto
2 t mul a3b a10b a11b a7b tOnSpline lineto% note the cross-over
2 t mul a3f a10f a11f a7f tOnSpline lineto
t       a0b a1b  a2b  a3b tOnSpline lineto
closepath
t       a7f a6f  a5f  a4f tOnSpline moveto
2 t mul a4b a9b  a8b  a0b tOnSpline lineto
2 t mul a4f a9b  a8f  a0f tOnSpline lineto
t       a7b a6b  a5b  a4b tOnSpline lineto
closepath}for}bind def %end dostringing
%%EndProlog
%
%---Program--- the script
%
/phi  30 def /theta 5 def /scalingfactor 50 def
reversevideo 1 setgray %white further on
3  setlinewidth frame        stroke
2  setlinewidth approxellipsestroke
.5 setlinewidth dostringing  stroke
showpage
%%EOF
```

Gabo has made variations of the LINEAR CONSTRUCTION No 1, where the sides of the frame are varied.

Appendix 4 Linear Construction in Space No 2 emulation in EPSF

A MetaFont program has been published in MAPS in 1996, where use is made of `path p[]`, `dir`, `down`, `left`, `hide`, `point ... of`, `for ... endfor`, `xpart`, `ypart`, `drawoptions(withcolor white)`, `draw`, `fill ... reverse ... cycle withcolor black`. Still a nice program after so many years. I paid attention this time also to the elliptic cut outs in the perspex frame in the yz slab as well as in the xz slab.

I had difficulties in passing along the circumference of the frame in equidistant steps: after `flattenpath` {}{}{}{}`pathforall` did not give me an equidistant set of points on the stack, because of too much change in curvature.

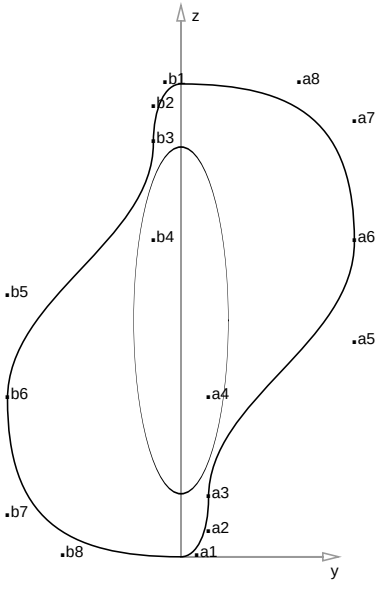
Moreover, MetaFont/MetaPost's part of the frame path

```
p1:= (0, 3size){right}..{down}(1.1size, 1.75size)..
     (.175size, .375size)..{left} origin;
```

I could not precisely transcribe into PostScript. How to mimic the directions `left`, `down` and ilks? Of course in the direction of the tangents of the curve at the point, but how far, taking into account scaling? Furthermore, in PostScript you can't apparently supply a point without the control points for drawing a B-cubic through the point.

The Frame of the object consists of a BarbaPapa-like shape in perspex, with an elliptic cutout, intersected with a copy rotated over 90° along the vertical axis.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Linear Construction in Space No 2, Frame in yz plane
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%CreationDate: Februari 2011
%%BoundingBox: -115 -15 125 355
%%BeginSetup
%%EndSetup
%%DocumentFonts: Helvetica
%%Pages: 0
%%EndComments
(C:\\PSlib\\Pslib.eps) run % used: origin arrow dotsandnames s  H12pt
/y-axis { origin  1 s    0   .1 5 10 arrow } def
/z-axis { origin    0  3.5 s   .1 5 10 arrow } def
 1 s    0    moveto -5 -12 rmoveto (y)  H10pt setfont show
   0  3.5 s  moveto  7 -10 rmoveto (z)                show
%yz plane
/a0 {0         0       } def
/a1 { .1    s  0       } def
/a2 { .175  s   .15  s } def
/a3 { .175  s   .375 s } def
/a4 { .175  s  1     s } def
/a5 { 1.1   s  1.35  s } def
/a6 { 1.1   s  2     s } def
/a7 { 1.1   s  2.75  s } def
/a8 {  .75  s  3     s } def
/a9 { 0        3     s } def
%mirrored and upside down, -yz plane
/b0 { 0       3     s } def
/b1 { -.1   s 3     s } def
/b2 { -.175 s 2.85  s } def
/b3 { -.175 s 2.625 s } def
/b4 { -.175 s 2     s } def
```



```
/b5 {-1.1   s 1.65  s } def
/b6 {-1.1   s 1     s } def
/b7 {-1.1   s  .25  s } def
/b8 {-.75   s 0     s } def
/b9 { 0       0       } def
%
/frame{
a0 moveto
a1 a2 a3 curveto
a4 a5 a6 curveto
a7 a8 a9 curveto
%b0 moveto
b1 b2 b3 curveto
b4 b5 b6 curveto
b7 b8 b9 curveto
closepath} def
%
/ellipse{gsave
   0 1.5 s translate
  .3 1.1 scale
   0 0 1 s 0 360 arc stroke
grestore}def
%%EndProlog
%
%Program
%
/scalingfactor 100 def
frame ellipse stroke
[(a1) (a2) (a3) (a4) (a5) (a6) (a7) (a8)
 (b1) (b2) (b3) (b4) (b5) (b6) (b7) (b8)] dotsandnames
y-axis z-axis  .6 setgray stroke
showpage
```

Stringing is more difficult.

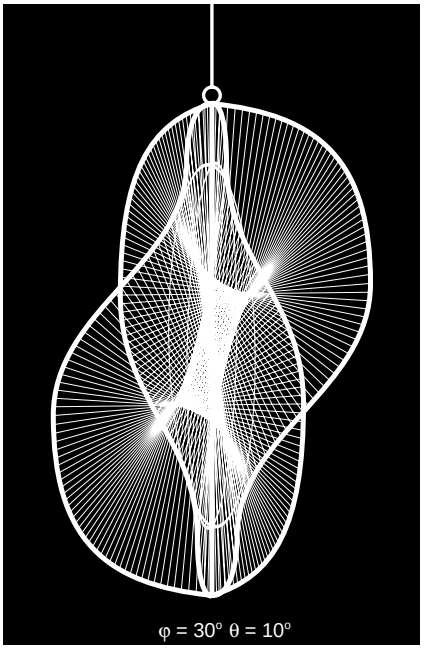
The origin is chosen at the foot of the object. The (unprojected) circumference in the first quadrant of the yz plane consists of three B-cubics, with varying curvature. Roughly equidistant points are sampled and stored on the stack. Each point is and projected and transformed into the $x^{-}z$ quadrant, i.e. the plane through the negative x axis and the positive z axis, and also projected. Both projected points are connected by a line (wire). Schematically, for each point on the stack

$$(0,\ y,\ z)\texttt{ptp<-string->}(-y,\ 0,\ size - z)\texttt{ptp} \quad \text{with} \quad size = 3s, s = 100.$$

This is repeated for the $y^{-}x^{+}$ quadrant, and programmed with the sample points on the stack (too low-levelish, yes I know, but ... interesting; for the other quadrants the sample values on the stack are copied into an array, for easy multiple access, not difficult at all).

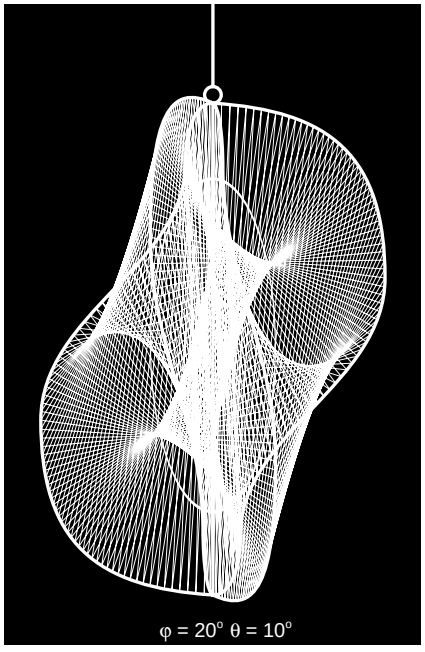
```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Emulation of Naum Gabo Linear Construction in Space No 2
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%CreationDate: Februari 2011
%%BoundingBox: -125 -30 125 355
%%ProgramFonts: Helvetica
%%BeginSetup
%%EndSetup
%%Pages: 0
%%EndComments
(C:\\PSlib\\Pslib.eps) run %used: origin dotsandnames s ptp tOnSpline
/reversevideo{-125 -30 250 385 rectfill}def%Mimics BoundingBox
/rope{0 355 moveto
      0 0 0 3 s ptp 5 add 5 90 450 arc}def %ring
%%EndProlog
%In yz plane data points for the frame
/a0 {0 0        0       ptp} def
```

```
/a1 {0  .1   s  0       ptp} def
/a2 {0  .175 s   .15  s ptp} def
/a3 {0  .175 s   .375 s ptp} def
/a4 {0  .175 s  1     s ptp} def
/a5 {0 1.1   s  1.35  s ptp} def
/a6 {0 1.1   s  2     s ptp} def
/a7 {0 1.1   s  2.75  s ptp} def
/a8 {0  .75  s  3     s ptp} def
/a9 {0  0       3     s ptp} def
%
%Roughly Equidistant Sampling of frame part in yz plane, with phi=0 and theta=0
/sampleframe{/phi 0 def /theta 0 def   %results on the operand stack
0 .175  .95{ a0 a1 a2 a3 tOnSpline }for
0.02 .028 .99{ a3 a4 a5 a6 tOnSpline }for
0  .025 1  { a6 a7 a8 a9 tOnSpline }for
}def
%rotated -xz plane data points for frame
/a0r { 0       0 0       ptp} def
/a1r {  .1   s 0 0       ptp} def
/a2r {  .175 s 0  .15  s ptp} def
/a3r {  .175 s 0  .375 s ptp} def
/a4r {  .175 s 0 1     s ptp} def
/a5r { 1.1   s 0 1.35  s ptp} def
/a6r { 1.1   s 0 2     s ptp} def
/a7r { 1.1   s 0 2.75  s ptp} def
/a8r {  .75  s 0 3     s ptp} def
/a9r { 0       0 3     s ptp} def
%mirrored and upside down, -yz plane data points for frame
/b0 {0  0       3     s ptp} def
/b1 {0  -.1   s 3     s ptp} def
/b2 {0  -.175 s 2.85  s ptp} def
/b3 {0  -.175 s 2.625 s ptp} def
/b4 {0  -.175 s 2     s ptp} def
/b5 {0 -1.1   s 1.65  s ptp} def
/b6 {0 -1.1   s 1     s ptp} def
/b7 {0 -1.1   s  .25  s ptp} def
/b8 {0 -.75   s 0       ptp} def
/b9 {0   0      0       ptp} def
%rotated -xz plane data points for frame
/b0r { 0       0 3     s ptp} def
/b1r { -.1   s 0 3     s ptp} def
/b2r { -.175 s 0 2.85  s ptp} def
/b3r { -.175 s 0 2.625 s ptp} def
/b4r { -.175 s 0 2     s ptp} def
/b5r {-1.1   s 0 1.65  s ptp} def
/b6r {-1.1   s 0 1     s ptp} def
/b7r {-1.1   s 0  .25  s ptp} def
/b8r { -.75  s 0 0       ptp} def
/b9r { 0       0 0       ptp} def
%
/frame{%executed with user projection angles
a0 moveto
a1 a2 a3 curveto
a4 a5 a6 curveto
a7 a8 a9 curveto
a0r moveto
a1r a2r a3r curveto
a4r a5r a6r curveto
a7r a8r a9r curveto
b0 moveto
b1 b2 b3 curveto
b4 b5 b6 curveto
b7 b8 b9 curveto
b0r moveto
b1r b2r b3r curveto
b4r b5r b6r curveto
```



```
b7r b8r b9r curveto
closepath
}def %end frame
%
/dostringing{count %83 number of samples x and y on the stack
/n exch def
n copy %for stringing 2 quadrants
%Stringing +yz<->-xz
n 2 div cvi {2 copy      % y z y z on stack
             0 3 1 roll  % y z 0 y z
             ptp         % y z projected point
             moveto      % y z
             exch neg exch% -y z
             3 s sub neg % -y 3s-z
             0 exch      % -y 0 3s-z
             ptp         % projected pair
             lineto
            }repeat
%Stringing +xz<->-yz
n 2 div cvi {2 copy      % y z y z on stack
             0 2 1 roll  % y z y 0 z
             ptp         % y z projected point
             moveto      % y z
             0 3 1 roll  % 0 y z
             exch neg exch% 0 -y z
             3 s sub neg % 0 -y 3s-z
             ptp         % projected pair
             lineto
            }repeat
%other quadrants
%stringing +xZ<->+yz
/np1 n 1 add def%n-1 last element of samples array
/cnt 0       def%first element of samples array
n 2 div cvi {/cnt cnt 2 add def
% in +yZ plane
0                       %x=0 inserted
samples n   cnt sub get %y value from samples array
samples np1 cnt sub get %z value
ptp moveto
% rotate to +xz plane
samples cnt 2 sub get %y value from samples array --> x
0                     %y=0 inserted
samples cnt 1 sub get %z value
ptp lineto
}repeat
%stringing -xz<->-yz
/np1 n 1 add def%n-1 last element of samples array
/cnt 0       def%first element of samples array
n 2 div cvi {
/cnt cnt 2 add def
% in +yZ plane
0                                  %x=0 inserted
samples n   cnt sub get neg        %y value from samples array
samples np1 cnt sub get 3 s sub neg %mirrored z value
ptp moveto
% rotate to +xz plane
samples cnt 2 sub get neg         %y value from samples array ->  x
0                                 %y=0 inserted
samples cnt 1 sub get 3 s sub neg %mirrored z value
ptp lineto
}repeat
} bind def %end dostringing
%
/yz-approxellipseNo2stroke{gsave%implicit in yz plane, phi, theta must be given
0 1.5 s translate
.3  1.1 scale%kind of ellipse
/a0 {0 -1   s  0     ptp} def
```

```
/a1 {0 -1   s  .55 s ptp} def
/a2 {0 -.55 s  1   s ptp} def
/a3 {0  0      1   s ptp} def
/a4 {0  .55 s  1   s ptp} def
/a5 {0  1   s  .55 s ptp} def
/a6 {0  1   s  0     ptp} def
/a7 {0  1   s -.55 s ptp} def
/a8 {0  .55 s -1   s ptp} def
/a9 {0  0     -1   s ptp} def
/a10{0 -.55 s -1   s ptp} def
/a11{0 -1   s -.55 s ptp} def
a0 moveto a1  a2  a3 curveto
          a4  a5  a6 curveto
          a7  a8  a9 curveto
          a10 a11 a0 curveto closepath stroke
grestore} def  %end yz-approxellipseNo2stroke
%
/xz-approxellipseNo2stroke{gsave%implicit in xz plane, phi, theta must be given
0 1.5 s translate
.3  1.1 scale%kind of ellipse
/a0 {-1   s 0  0     ptp} def
/a1 {-1   s 0  .55 s ptp} def
/a2 {-.55 s 0  1   s ptp} def
/a3 { 0     0  1   s ptp} def
/a4 { .55 s 0  1   s ptp} def
/a5 { 1   s 0  .55 s ptp} def
/a6 { 1   s 0  0     ptp} def
/a7 { 1   s 0 -.55 s ptp} def
/a8 { .55 s 0 -1   s ptp} def
/a9 { 0     0 -1   s ptp} def
/a10{-.55 s 0 -1   s ptp} def
/a11{-1   s 0 -.55 s ptp} def
a0 moveto a1  a2  a3 curveto
          a4  a5  a6 curveto
          a7  a8  a9 curveto
          a10 a11 a0 curveto closepath stroke
grestore} def  %end xz-approxellipseNo2stroke
%
%---Program--- the script
%
/scalingfactor 100 def
reversevideo 1 setgray %white lines, further on
sampleframe %with phi=theta=0!
/phi 30 def /theta 10 def% viewing angle
.5 setlinewidth dostringing stroke
3  setlinewidth frame       stroke
2  setlinewidth yz-approxellipseNo2stroke
                xz-approxellipseNo2stroke
                rope        stroke
showpage
%%EOF %Of my most complete Linear Construction in Space No 2 emulation
```



The .eps result from MetaPost comprises 800+ lines and is, alas, not intelligible. I can't even recognize the created path.

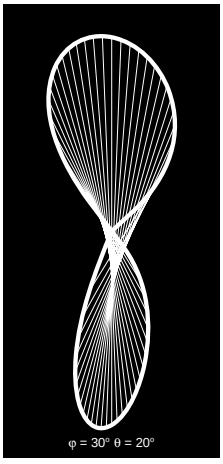
Note the flexible way of specifying a path in MetaFont/MetaPost. In PostScript we can just construct piecewise B-cubics. A general curve is approximated after projection of the sample points by `lineto's`. Of course we can write procedures of our own, to imitate MetaPost's flexibility. Another difference with respect to `hidden lines' is that MetaPost has an `undraw` macro while in PostScript we must explicitly draw in the colour of the background.

Gabo has made LINEAR CONSTRUCTION IN SPACE No 2 in 1949, 1958, 1962-64, 1972-73, and 1976.

My 1996, simplified, pseudo-animated version was on my WWW of old, and is included below. It does not show all lines; nevertheless nice in its simplicity.

Variations

Gabo has made variations among others based on an octahedral frame. I made a variation with a twisted lemniscate frame —lemniscate in polar coordinates $r^2 = 2a^2 \cos 2\theta$ — in an hour or two. The octahedral variant would take another hour to emulate.



The architectural variation, he never realized, was a commission for the Esso Company NY: above each of 2 revolving entrance doors a `Linear Construction No 2` revolving at a slower speed than the doors.

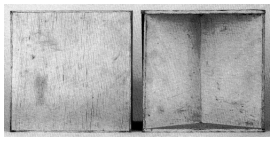
The wall between was a relief in itself: a set of perspective lines that seem to break the wall into a deeper space, and within these lines, a large construction of spiralling arcs around a vertical axis.

Appendix 5 Emulation of a Spheric Theme in EPSF

Spheric Themes demonstrate another way of how Gabo constructed (the idea of) surfaces: by planes orthogonal to the surface.

He called it the stereometric method of representing surfaces.

> Stereometry is a branch of mathematics concerned with the description of 3D objects and calculations related to these. For Gabo a cube is not represented by its 6 surfaces, but by top, bottom, and the intersecting diagonal planes, making the inside visible



Famous are his constructed head series, such as the earlier shown `Head in a Corner Niche`, and the `Head ...` on the book cover. In the Spheric Theme we are about to emulate both ways of representing surfaces: stereometric and stringed surfaces are integrated in one object.

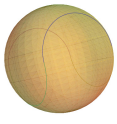
Gabo invented how to model his Spheric Theme curves:

> `The basic form was made by taking two identical flat pierced circles or broad rings and making a single cut in each along the line of a radius. The two discs are bent in a serpentine curve and butt-jointed to each other at both ends. The resulting figure fits exactly into a sphere and the outer edges of the discs form the interlocking curves like those that divide the pieces of felt covering the tennis ball.'

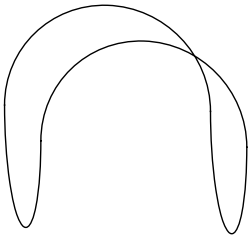
Frame

The outer circumference of the frame is the curve on the tennis ball, a little deformed. The four parts of the tennis ball curve are given by the formulas

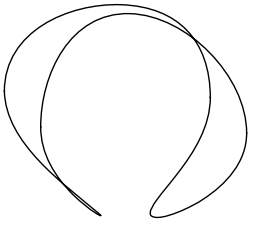
$$\begin{aligned}&\{(x,y,z)|(1,\sin t,\cos t)\}\\&\{(x,y,z)|(-1,\sin t,\cos t)\}\\&\{(x,y,z)|(-\sin t,1,-\cos t)\}\\&\{(x,y,z)|(-1,-\sin t,-\cos t)\}\end{aligned}\qquad t\in(-\frac{\pi}{2},\frac{\pi}{2})$$



I modified the tennis ball curve by the deformation $1 \to (1-d)\sin^2 t + d$, $d = .25$, see figure below, which comes close to Gabo's. For the inner circumference I took the outer scaled by a factor 3.

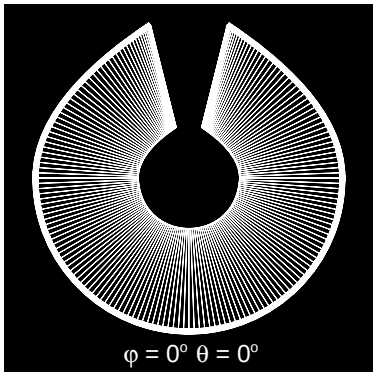


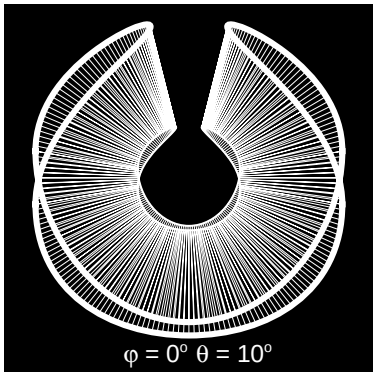
Curve Tennis ball $\varphi = \theta = 10^o$

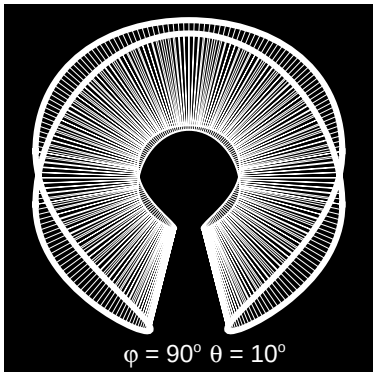


Mod: 1 -> .75sin^2(t)+.25, $\varphi = \theta = 10^o$

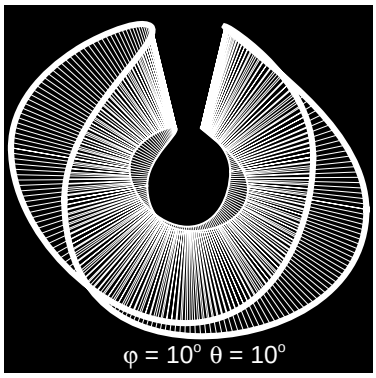
```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -90 -90 90 90
%%BeginSetup
%%EndSetup
%%Title: Emulation of Naum Gabo Spheric Theme
%%Creator: Kees van der laan, kisa1@xs4all,nl.
%%CreationDate: feb 2011
%%DocumentFonts: Helvetica Symbol
%%EndComments
(C:\\PSlib\\PSlib.eps) run %used: s ptp
%
/reversevideo{-90 dup 180 dup rectfill}def %Mimics the BoundingBox
%
/sphericthemestroke{%
%%%part upper front x>0 {t|(ft, sin t, -cos t)}, f(-90)=1
1 s   -1 s    0 ptp moveto
-89 1 90{/t exch def
ft s   t sin s   t cos s neg ptp lineto
}for stroke
%%%part upper back x<0 {t|(-ft, sin t, -cos t)}
-1 s   -1 s   0  ptp moveto
-89 1 90{/t exch def
ft neg s   t sin s   t cos s neg  ptp lineto
}for stroke
%%%part right y>0 {t|(-sin t, ft,  cos t)}
1 s   1 s   0 ptp moveto
-89 1 90{/t exch def
t sin neg s   ft s   t cos  s    ptp lineto
}for stroke
%%%part left y<0 {t|(-sin t, -ft,  cos t)}
1 s   -1 s   0 ptp moveto
-89 1 90{/t exch def
t sin neg s    ft neg  s    t cos s   ptp lineto
}for stroke
} bind def %end sphericthemestroke
%
/dostringingstroke
{%%%part upper front x>0 {t|(ft, sin t, -cos t)}, f(-90)=1
/v {.333 mul} def /step 2 def% implies number of strings
-90  step 90{/t exch def
ft s     t sin s    t cos s   neg ptp moveto
ft s v   t sin s v  t cos s v neg ptp lineto
}for
stroke
%%%part upper back x<0 {t|(-ft, sin t, -cos t)}
-90 step 90{/t exch def
ft neg s    t sin s    t cos s   neg ptp moveto
ft neg s v  t sin s v  t cos s v neg ptp lineto
}for
stroke
%%%part right y>0 {t|(-sin t, ft,  cos t)}
```







```
-90 step 90{/t exch def
t sin neg s     ft s     t cos s     ptp moveto
t sin neg s v   ft s v   t cos s v   ptp lineto
}for
stroke
%%%part left y<0 {t|(-sin t, -ft,  cos t)}
-90 step 90{/t exch def
t sin neg s     ft neg  s     t cos s    ptp moveto
t sin neg s v   ft neg  s v   t cos s v  ptp lineto
}for
stroke
} bind def %end dostringingstroke
%%EndProlog
%
%---Program--- the script
%
reversevideo 1 setgray /d .25 def
/ft { 1 //d sub t sin dup mul mul //d add} def
/phi 90 def /theta 10 def
/scalingfactor 25 def sphericthemestroke
/scalingfactor 75 def sphericthemestroke
.1 setlinewidth dostringingstroke
showpage
%%EOF
```

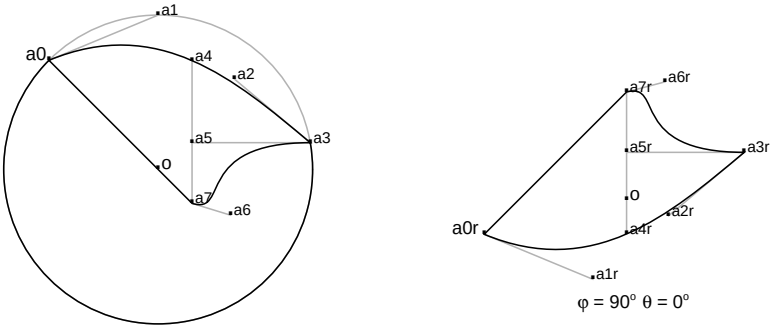


Appendix 6 Linear Construction in Space: Suspended

One of Gabo's favorites, which he used to display on retrospective exhibitions.

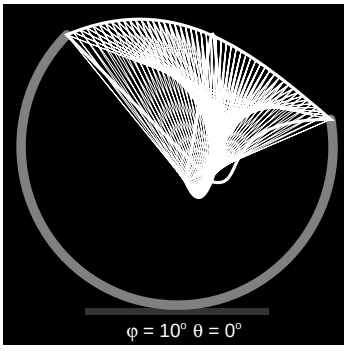
Frame The frame consists of 2 identical, `triangular' slabs of clear plastic orthogonal (rotated and mirrored) to each other, suspended on a metal arc with a wooden base.

What motivated Gabo to create just these triangular shapes? What were the restrictions? Did the balance of the object play a role, in choosing the rotation symmetry axes? I don't think so, just his feeling for beauty, I guess. The unbalance creates tension.



Left the frame part in the yz plane, right the (rotated and mirrored) frame part in the xz plane. The frame part in the yz plane is realized by 2 splines: `a0 moveto a1 a2 a3 curveto a5 a5 a7 curveto closepath`.

For the first B-cubic the data points are a_0,a_3 with control points a_1,a_2. For the second B-cubic the data points are a_3,a_7 with control points a_5,a_6; not critical. The axis of rotation is the line from a_4 to a_7. The suffix `r` denotes the mirrored and rotated triangle points.



I did not include the program, because the rotation required details, which I expect too boring for the casual reader.

Appendix 7 Torsion (interactive) MetaFont program of old

For the MetaFont aficionados: the program below still works on my PowerMac. Not so difficult to adapt for those fluent in MetaPost, I presume.

```
%Gabo's torsion example. December 1995, CGL
%
tracingstats:=proofing:=1;screenstrokes;
"Torsion from different viewpoints";
string yorn;
message "Gabo's Torsion.";
message "Do you wish simplest variant? (y or n):";
yorn:= readstring;
size=50;
def openit = openwindow currentwindow
   from origin to (screen_rows,screen_cols) at (-size,300)enddef;
pickup pencircle scaled .005pt;
pair aux[];
path  po[], pi[];
if yorn="n": d=.1size
        else: d=0 fi;
r=.2size;
for b= 20:%15step10until45:
for a= 0 step30until90:
```

```
currentpicture:=currentpicture shifted (2size,0);
%The following def must be included or a, b, must be supplied as arguments.
def ptp(expr x,y,z)=(-x*cosd a +y*sind a, -x*sind a *sind b -y*cosd a *sind
b+ z*cosd b)enddef;
po1:=ptp(0,-size,2d)--ptp(0,0,size+2d)--ptp(0,size,2d)&
     ptp(0, size,2d)..ptp(0,0,0)..ptp(0,-size,2d)..cycle;
aux0:=.5[ptp(0,-size,2d),ptp(0,0,size+2d)];
aux1:=.5[ptp(0,size,2d),ptp(0,0,size+2d)];
pi1:=ptp(0,-size+2.5d,2.5d)..controls aux0..
    ptp(0,0,size-d)..controls aux1..
    ptp(0,size-2.5d,2.5d)...
    ptp(0, size-2.5d,2.5d)...ptp(0,0,d)...ptp(0,-size+2.5d,2.5d)..cycle;
po2:=ptp(-size,0,-2d)--ptp(0,0,-size-2d)--ptp(size,0,-2d)&
     ptp(size,0,-2d)..ptp(0,0,0)..ptp(-size,0,-2d)..cycle;
aux3:=.5[ptp(-size,0,-2d),ptp(0,0,-size-2d)];
aux4:=.5[ptp(size,0,-2d),ptp(0,0,-size-2d)];
pi2:=ptp(-size+2.5d,0,-2.5d)..controls aux3..
    ptp(0,0,-size+d)..controls aux4..
    ptp(size-2.5d,0,-2.5d)...
    ptp(size-2.5d,0,-2.5d)...ptp(0,0,-d)...
    ptp(-size+2.5d,0,-2.5d)..cycle;
%Foot
po3:=ptp(r, 0,-size-2d)..ptp( .706r, -.706r,-size-2d)..
     ptp(0,-r,-size-2d)..ptp(-.706r, -.706r,-size-2d)..
     ptp(-r,0,-size-2d)..ptp(-.706r,  .706r,-size-2d)..
     ptp( 0,r,-size-2d)..ptp( .706r,  .706r,-size-2d)..cycle;
%
if yorn="n":fill po1; unfill pi1;fill po2; unfill pi2
            ;draw ptp(0,0,-size+d)--ptp(0,0,-size-2d)
            ;draw ptp(0,0,size-d)--ptp(0,0,size+2d)
        else:draw po1; draw po2; draw pi1; draw pi2 fi;
fill po3;
for k=0 upto 20:
 draw point .1k of pi1--point 5-.1k of  pi2;
endfor
for k=0 upto 20:
 draw point 3+.1k of pi1--point 2-.1k of  pi2;
endfor
showit;
endfor
endfor
end
```