contextgroup

Proceedings

6° ConTEXt meeting, Breskens, The Netherlands

Imprint

At the fourth ConTEXt meeting in the Czech Republic, the idea came up to create an organisation that would foster ConTEXt development and act as a showcase to attract new users and support the old ones, much like TUG and the local user groups helped shaping and promoting TEX in the 1980s.

That group now exists formally with the name Context Group, as a non-profit organisation in French law ("association loi 1901"). A number of people have volunteered to be on the board, which held its first meeting during the 5th ConTEXt Meeting. We hope to set up an organisation that will inspire people to contribute even more to the ConTEXt community, and we need your help!

The Context Group offers its members:

- A yearly multi-day ConTEXt meeting somewhere in Europe.
- This journal, which also doubles as the proceedings of that meeting.
- An easy to install and update ConTEXt installation.
- A discount to other TEX related conferences.

Becoming a member is possible by filling in the form on the Context Group website (URL on the left) and paying the membership fee via paypal or bank transfer. The yearly membership fee is currently 40 € for regular individuals and 20 € for students.

Articles can be sent to `cg-journal@contextgarden.net`, preferably using the `cgj` module which can be found in the ConTEXt distribution.

This journal is typeset using the latest ConTEXt beta at the time of production, using the latest version of LuaTEX. The used fonts are the commercial 'Alwyn New' family by Chris Dickinson of *moretype* for the running text, the freeware font 'Inconsolata' for code blocks, and 'TEX Gyre Pagella math' for mathematics.

Contents

Dayplan

Monday, 8 October 2012

09:00		Conference opening
09:15	Kees van der Laan	Recreational TEX&Co - with a serious undertone
10:15	Jano Kula	Run for Fun
11:30	Mari Voipio	TEXtile crafts
14:00	Kees van der Laan	Julia fractals in PostSCript
14:45	Taco Hoekwater	MetaClock
16:00	Patrick Gundlach	Database publishing with LuaTEX and the speedata Publisher
16:45	Patrick Gundlach	A journey to the land of LuaLATEX
17:30		Dante membership meeting
20:30		CG membership meeting

Tuesday, 9 October 2012

09:00	Uwe Ziegenhagen	Professional Business Reports with LATEX
09:45	Leo Arnold	Many versions from one source - LATEX for lecturers and teachers
11:00	Taco Hoekwater	MetaPost path resolution isolated
11:45	Taco Hoekwater	Parsing PDF content streams with LuaTEX
14:00	Hans Hagen	context: the script
14:45	Hans Hagen	context: after the cleanup
16:00	Luigi Scarso	MFLua
16:45	Mari Voipio	Metapost workshop
20:30	Willi Egger	Conference folder workshop

Wednesday, 10 October 2012

09:00		Excursion
19:30		Conference Dinner

Thursday, 11 October 2012

09:00	Bogusłav Jackowski	OTF math fonts: GUST e-foundry's workbench
09:45	Jerzy Ludwichowski	Present and future of the TG Math Project: the report and some questions
11:00	Piotr Strzelczyk	Is backward compatibility of LM Math and CM math sensible?
11:45	Gust Foundry	BOF session: 'The future of math fonts'
14:00	Hans Hagen	xml
14:45	Hans Hagen	a couple of styles
16:00	Hans Hagen	lexing
16:45	Hans Hagen	(visual) debugging

Friday, 12 October 2012

09:00	Yamamoto Munehiro	TEX Typesetting Circumstances for Japanese Publishing
09:45	Kitagawa Hironori	Japanese Typesetting with LuaTEX
11:00	Hans Hagen	Tricks with the parbuilder (Arabic typesetting)
11:45	Ivo Geradts Kai Eigner	Typesetting Sanskrit with LuaTEX
14:00	Hans Hagen	mixed columns
14:45	Tomás Hála	Differences in typesetting rules between Czech and Slovak languages (in the context of ConTEXt)
16:00	Jean-Michel Hufflen	MlBibTEX and Its New Extensions
16:45	Jean-Michel Hufflen	Demonstration of the `mlbibcontext` Program
17:45	Sietse Brouwer	Bof session: Context Wiki
18:30		2013 Announcements
18:45		Conference closing

CrafTEX
Applying TEX and friends in crafts
Mari Voipio

Everything started at my job as Documentation manager in hi-tech industry: when the word processor gave up on our big fat instruction manual and the purpose-built software wasn't within budget, we ended up with ConTEXt. The transit period wasn't easy, but in time I learned to appreciate this software that does *not* make assumptions on what I'm attempting to do.

Thus, when I suddenly found myself with a textile craft book to be translated and prepared for printing, I thought of ConTEXt. Life happened and the booklet is still sitting on my desk waiting for its turn, but in the meanwhile I have learned many other things about TEX based systems and started to exploit their potential in crafts.

The experience has been mind-blowing! I come up with new implementations almost weekly, although I don't usually try things out until I have a real need for them. I am not a programmer, but I realize that a computer is especially useful in reducing the tedious repetitiveness of the planning stages. Nothing can ever prepare a crafter to what happens when you play with real yarn and real paper and glue, but the "what if that's lighter blue" and "I guess this really is wrong font here" process can be speeded up significantly by computer simulation.

I don't feel complete until I've shared my knowledge with others. I don't get that many face-to-face opportunities to that, so I decided to go online instead, http://www.lucet.fi/. I haven't had the energy to fight with WordPress about the printing styles, so instead I'm planning to do printer-friendly pdf instructions with ConTEXt and MetaPost.

Besides enhancing my creativity, I also use ConTEXt to deal with the boring but necessary parts of having my own little craft business, e.g. for creating price lists and business cards. This migration is still very much in process, but eventually everything will be done with ConTEXt and possibly MetaPost and with as few different style definitions as possible.

MetaPost: PNG Output

Taco Hoekwater

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

Introduction

For one of my presentations at EuroTEX2012 in Breskens, I wanted to create an animation in order to demonstrate a Metapost macro that uses timer variables to progress through a scene.
While working on that presentation, it quickly became obvious that the 'traditional' method of creating an animation with Metapost by using ImageMagick's `convert` to turn EPS images into PNG images was very time consuming. So much so, that I actually managed to write a new backend for Metapost while waiting for ImageMagick to complete the conversion.

Simple usage

Metapost will create a PNG image (instead of Encapsulated PostScript or Scalable Vector Graphics) by setting `outputformat` to the string png:

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

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

Adjusting the bitmap size

In the simple example given above, Metapost has used the default conversion ratio where one point equals one pixel. This is not always desired, and it is tedious to have to scale the picture whenever a different output size is required. To allow easy modification of the bitmap size independent of the actual graphic, two new internal parameters have been added: hppp and vppp (the names come from Metafont, but the actual meaning is specific to Metapost).
In Metapost, 'hppp' stands for 'horizontal points per pixel', and similarly for 'vppp'. Adding

```
hppp := 2.0;
```

to the example above changes the bitmap into 50 x 100 pixels. Specifying values less than 1.0 (but above zero!) makes the bitmap larger.

Adjusting the output options

Metapost creates a 32-bit RGBA bitmap image, unless the user alters the value of another new internal parameter: `outputformatoptions`.
The syntax for `outputformatoptions` is a space-separated list of settings. Individual settings are *keyword* + = + *value*. The only currently allowed ones are:

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

No spaces are allowed on the left, nor on the right, of the equals sign inside a setting.
The assignment that would match the compiled-in default setup is:

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

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

Some notes on the different PNG output formats:

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

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

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

  $$r = 1 - (c + k > 1 ? 1 : c + k);$$
  $$g = 1 - (m + k > 1 ? 1 : m + k);$$
  $$b = 1 - (y + k > 1 ? 1 : y + k);$$

  If you care about color conversions, you should be doing a `within <pic>` loop inside `extra_endfig`. The built-in conversions are more of a fallback.

What you should also know

Metapost uses cairo (http://cairographics.org) to do the bitmap creation, and then uses libpng (http://www.libpng.org) to create the actual file. Any `prologues` setting is always ignored: the internal equivalent of the `glyph of` operator is used to draw characters onto the bitmap directly.
If there are points in the current picture with negative coordinates, then the whole picture is shifted upwards to prevent things from falling outside the generated bitmap.

Database publishing with LuaTEX and the speedata Publisher

Patrick Gundlach

Database Publishing is the repetitive (semi) automatic transformation from a data source to some kind of output (HTML, PDF, epub,...). A common task is to have an excel sheet, a product information management system or a webshop database and generate reports, data sheets, product catalogs or other kind of PDF documents out of it. Database publishing is often equal to "InDesign Publishing" with the help of some plugin that automates the task of pulling data from the database into the document. The user can (and must) make the resulting document more beautiful.

There are several alternatives to this approach, especially when you need 100% unattended workflows. Each alternative has advantages and of course drawbacks. 1) ConTEXt fills this gap nicely, but requires a very knowledgable programmer. 2) Many times users write some perl or python scripts that reads the database contents and produces some kind of output, perhaps LaTEX code that must be run with PdfLaTEX. This is a fast approach, but tend to get very hackerish after some time. 3) There is a standardized way of transforming XML to PDF called XSL-FO. This w3c standard has the big advantage that many tools exist to help the user in the task of publishing. But XSL-FO is very limited in its capabilities to produce reasonable documents.

A common demand in high volume output is to optimize page usage. As an example: image you have six products in a group but a page only fits five. The software system should be able to re-arrange the products and change a few parameters (image size, text size, text length), so that all six products fit on the same page and thus a whole page saved. The aforementioned systems are either very demanding on the programming side or just not capable of optimizations like these.

The product of our company is filling in this gap. It provides a way to transform XML (and thus any data) to PDF. It has a specialized input language designed for the purpose of laying out elements on a page, and it has all functionality of a modern programming language (variables, loops, if-then-else switches). It can put text and graphical elements on a virtual page that is used for any kind of layout optimization. These virtual pages can be removed and re-typeset with different parameters and only the ‘best’ page will make it to the PDF. As there is no control language for this kind of application yet, the system is inspired by the standards HTML (table layout), XPath (accessing the data and running specialized functions) and XSLT (accessing document nodes, programming constructs).

The software (called ‘speedata Publisher’) is written in Lua and makes heavy use of the LuaTEX engine. We use TEX to break paragraphs into lines, arrange the programmatically created boxes and glue for layout of complex tables and to write clean PDF. The publisher is open source software (AGPL) and runs under the three major operating systems (Linux, Windows, Mac OS X). The documentation is mostly still in German, although we are currently translating the documentation into english.

Minutes of the 2nd ConTEXt Group membership meeting

Willi Egger

Present	Bernd Militzer, Frans Goddijn, Hans Hagen, Harald König, Jano Kula, Jean-Michel Hufflen, Luigi Scarso, Mari Voipio, Martin Schröder, Thomáš Hala, Taco Hoekwater, Willi Egger,
Excused	John Haltiwanger, Mojca Miklavec, Thomas Schmitz, Wolfgang Schuster
Absent	Adelheid Grob, Peter Münster
Guest	Yusuke Kuroki

1. Opening

The meeting started at 20:33.

2. Agenda

1. Opening
2. Agenda
3. Activities since last meeting
4. Board
5. Financials
6. Design of the corporate identity
7. Journal/Proceedings
8. Next projects
9. Questions by members
10. Closing

3. Activities since last year

The main things achieved during the past period is the setup and make running of the bank account. We have now a bank account opened in France. The treasurer and the president have access to it and can perform actions through a web-interface. There is also a PayPal account to ease payment with foreign currencies. In order to get the bank affairs right we were obliged to open the account in France after having tried to do so in Germany, The Netherlands and Slovenia.

Another activity is the development of a corporate identity for the CG-group. We come back to this in Agenda point 6.

4. Board

Shortly after the last meeting the board has changed. As discussed during the last meeting Sébastien Mengin is followed up by Peter Münster. The reason is due to the requirements for opening a bank account in France. The best is that this is at the address of the association and therefore Peter was asked whether he could take up this duty.
Arthur Reutenauer resigned from the board on his own request.
Our formal treasurer does not respond to the board-mailing list. We found Mojca Miklavec prepared to take the function of treasurer. The collected money at CM5 is handed over to Mojca and after opening the account transferred to the bank. The gathering agrees to ask Adelheid to resign form the group's board.
None of the present members is willing to join the board at this moment.

Action	**Description**
Willi	Since the board has changed an announcement to the French authorities is required.
Willi	Ask Adelheid whether she wants to stay/resign from the board.

5. Financials

The treasurer has made up a summation of the financial situation, which is handed out at the meeting. There is a wrong figure in the totals-section. – This will be corrected.
Invoices over 2011 are submitted to the members. Still not all of them are payed. The invoices over 2012 are to be sent soon.

Overall the financial situation is ok, though we need to build on making some more capital. We will however easily be able to pay the expenses to come.
The financial report will be available from the CG-website.

Action	**Description**
Willi, Mojca	Fix the amount in the summation section.

6. Corporate Identity

We can proudly announce that we have found Adrian Egger (art-director in the graphic industry) willing to help us in establishing a corporate identity. We have styles made in ConTEXt for invoices, formal letters, reports like minutes of gatherings, and two types of envelopes.
In due time the style-guide will be released.
There is also a design for our journal. This style is not finalized completely, but will be finalized for the production of our first journal.

Action	**Description**
Willi, Adrian	Finalize the style-guide.

7. Journal/Proceedings

Due to the circumstances we have not yet our first journal containing the proceedings of CM5. The board decided during the board-meeting before the members meeting to make our first journal a combination of 5CM and 6CM. The gathering seconds this idea.

Action	**Description**
Adrian, board	Finalize the journal design and cover by **1st of november 2012**.
Writers, redaction	Collect all articles by **1st of november 2012**.

The planning is that the journal will go for production by the end of november. This will allow that the journal is sent to the members before X-mas.

8. Next projects

- **Website**: We need to ask Adrian to make a design of our new website, where all material concerning the GC-group is presented in a coherent way.
- **Archive for cartoons** We will implement an electronic archive for the cartoon coming from Duane Bibbi. We have already a couple of them and we want to install a tradition that the most active member of the group will be awarded with a cartoon for his work.
- This year's user of the year will be Sietse Brouwer, who invested a great effort in revisiting the wiki.
- **Next year's meeting**: Jano confirms, that the meeting will be held again in Czech Republic in combination with TEXperience. The place will be Brejlov. At the end of the EuroTEX 2012 the date is confirmed for week 39 which is the last full week of september.
- **Patricks's machine** is phased out within one month (end of oktober). First steps to safe the still running services have been already started up. The domain-server is moved to a machine at elvenkind (Taco) and Taco will be the responsible person for it.
  From november onwards the group is going to pay also the involved fees.
- ConTEXt-suite: Taco and Mojca make sure that there is a mirroring-system so that the suite is always available.

Action	**Description**
Taco	Take care of the shutdown of Patrick's machine, domain server moving.
Taco, Mojca	Setup a mirroring system for the ConTEXt-suite.

9. Questions by members

Mari thinks that it is correct to add the costs of a PayPal payment to the fee of membership. The gathering seconds this.

Action	Description
Mojca	Find out the average fee for transactions with PayPal and add it to the membership fee accordingly.

10. Closing

The meeting is closed at 21:22

MetaPost path resolution isolated

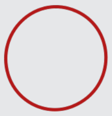
Taco Hoekwater

A new interface in MPLib version 1.800 allows one to resolve path choices programmatically, without the need to go through the MetaPost input language.

Metapost path solving

As we all know, MetaPost is pretty good at finding pleasing control points for paths. What all of you may know is that besides drawing on a picture, MetaPost can also display the found control points in the log file.
Some illustration at this point is useful. Here is the MetaPost path input source of a very simple path (as well as a visualisation of the path):

```
tracingchoices := 1;
path p;
p := (0,0) ..(10,10) ..(10,-5) ..cycle;
```



And here is what MetaPost outputs in the log file:

```
Path at line 5, before choices:
(0,0)
 ..(10,10)
 ..(10,-5)
 ..cycle

Path at line 5, after choices:
(0,0)..controls (-1.8685,6.35925) and (4.02429,12.14362)
 ..(10,10)..controls (16.85191,7.54208) and (16.9642,-2.22969)
 ..(10,-5)..controls (5.87875,-6.6394) and (1.26079,-4.29094)
 ..cycle
```

A more complex path of course creates more output, so:

```
p := (0,0)..(2,20){curl1}..{curl1}(10, 5)..controls (2,2) and
(9,4.5)..
  (3,10)..tension 3 and atleast 4 .. (1, 14){2,0} .. {0,1}(5,-4) ;

end.
```

produces:

```
Path at line 7, before choices:
(0,0){curl 1}
 ..{curl 1}(2,20){curl 1}
 ..{curl 1}(10,5)..controls (2,2) and (9,4.5)
 ..(3,10)..tension 3 and atleast4.5
 ..{4096,0}(1,14){4096,0}
 ..{0,4096}(5,-4)

Path at line 7, after choices:
(0,0)..controls (0.66667,6.66667) and (1.33333,13.33333)
 ..(2,20)..controls (4.66667,15) and (7.33333,10)
 ..(10,5)..controls (2,2) and (9,4.5)
 ..(3,10)..controls (2.34547,10.59998) and (0.48712,14)
 ..(1,14)..controls (13.40117,14) and (5,-35.58354)
 ..(5,-4)
```

But what if …

But what if you want to use that functionality outside of MetaPost, for instance in a C program?

> You will have to compile MPLib into your program; then create a Metapost language input string; execute it; and parse the log result.

All of that is not very appealing. It would be much better …

> if you could compile MPLib into your program; create a path programmatically; and then run the Metapost path solver directly; automatically updating the original path.

And that is what the current version of MPLib will allow you to do.

How it works

Once again, it is easiest to show you what to do by using a source code example:

```
#include "mplib.h"

int main (int argc, char ** argv)  {
        MP mp ;
        MP_options * opt = mp_options () ;
        opt -> command_line = NULL;
        opt -> noninteractive = 1 ;
        mp = mp_initialize ( opt ) ;
        my_try_path(mp);
        mp_finish ( mp ) ;
        free(opt);
        return 0;
}
```

Most of the example code above is exactly what one needs to do anything with MPlib programmatically. The only new line is the line calling `my_try_path(mp)`:

```
void my_try_path(MP mp) {
        mp_knot first, p, q;
        first = p = mp_append_knot(mp,NULL,0,0);
        q = mp_append_knot(mp,p,10,10);
        p = mp_append_knot(mp,q,10,-5);
        mp_close_path_cycle(mp, p, first);
        if (mp_solve_path(mp, first)) {
            mp_dump_solved_path(mp, first);
        }
        mp_free_path(mp, first);
}
```

This function uses a new type (`mp_knot`) as well as a bunch of new library functions in MPlib that exist since version 1.800.

- `mp_append_knot()` creates a new knot, appends it to the path that is being built, and returns it as the new tail of the path.
- `mp_close_path_cycle()` mimics `cycle` in the Metapost language.
- `mp_solve_path()` finds the control points of the path. `solve_path` does not alter the state of the used MPlib instance in away, it only modifies its argument path.
- `mp_dump_solved_path()` *user defined function, see below for its definition*
- `mp_free_path()` releases the used memory.

mp_dump_solved_path uses even more new functions. First let us look at its definition:

```
#define SHOW(a,b) mp_number_as_double(mp,mp_knot_##b(mp,a))
void mp_dump_solved_path (MP mp, mp_knot h) {
    mp_knot p, q;
    p = h;
    do {
        q = mp_knot_next(mp,p);
        printf ("(%g,%g)..controls (%g,%g) and (%g,%g)",
                SHOW(p,x_coord), SHOW(p,y_coord), SHOW(p,right_x),
                SHOW(p,right_y), SHOW(q,left_x), SHOW(q,left_y));
        p = q;
        if (p != h || mp_knot_left_type(mp,h) != mp_endpoint)
            printf ("\n ..");
    } while (p != h);
    if (mp_knot_left_type(mp,h) != mp_endpoint)
        printf("cycle");
    printf ("\n");
}
```

Somewhat hidden in the source above is that there is another new type: mp_number, the data structure representing a numerical value inside MPlib.

The used MPlib library functions are as follows:

- mp_knot_next() move to the next knot in the path.
- mp_knot_x_coord(), mp_knot_y_coord(), mp_knot_right_x(), mp_knot_right_y(), mp_knot_left_x(), mp_knot_left_y()
  return the value of a knot field, as a mp_number object (the calls to these functions are hidden inside the definition of the SHOW macro).
- mp_knot_left_type() returns the type of a knot, normally either mp_endpoint or mp_open.
- mp_number_as_double() converts a mp_number to double.

To satisfy our curiosity, here is the actual output of the example program listed above:

```
(0,0)..controls (-1.8685,6.35925) and (4.02429,12.1436)
 ..(10,10)..controls (16.8519,7.54208) and (16.9642,-2.22969)
 ..(10,-5)..controls (5.87875,-6.6394) and (1.26079,-4.29094)
 ..cycle
```

And that is almost exactly the same as in the log file:

```
(0,0)..controls (-1.8685,6.35925) and (4.02429,12.14362)
 ..(10,10)..controls (16.85191,7.54208) and (16.9642,-2.22969)
 ..(10,-5)..controls (5.87875,-6.6394) and (1.26079,-4.29094)
 ..cycle
```

The output is not perfectly the same because MetaPost itself does not use `mp_number_as_double` and `%g` for printing the `scaled` values that are by default used to represent numerical values.
The difference is not really relevant, since any programmatic use of the path solver should not have to be 100% compatible with the MetaPost programming language.

More complex paths

Of course there are also new functions to the create more complex paths that make use of `curl`, `tension` and/or `direction` specifiers.
Here is how to encode the second MetaPost path from the earlier example:

```
first = p = mp_append_knot(mp,NULL,0,0);
q = mp_append_knot(mp,p,2,20);
p = mp_append_knot(mp,q,10,5);
if (!mp_set_knotpair_curls(mp, q,p, 1.0, 1.0))
    exit ( EXIT_FAILURE ) ;
q = mp_append_knot(mp,p,3,10);
if (!mp_set_knotpair_controls(mp, p,q, 2.0, 2.0, 9.0, 4.5))
    exit ( EXIT_FAILURE ) ;
p = mp_append_knot(mp,q,1,14);
if (!mp_set_knotpair_tensions(mp,q,p, 3.0, -4.0))
    exit ( EXIT_FAILURE ) ;
q = mp_append_knot(mp,p,5,-4);
if (!mp_set_knotpair_directions(mp, p,q, 2.0, 0.0, 0.0, 1.0))
   exit ( EXIT_FAILURE ) ;
mp_close_path(mp, q, first);
```

Elaborate documentation for these extra functions (and a few more) is in `mplibapi.tex` which is included in the MetaPost distribution.

Lua interface

There is also a Lua interface for use in LuaTEX, which is a bit higher-level

```
<boolean> success = mp.solve_path(<table> knots, <boolean> cyclic)
```

This modifies the `knots` table (which should contain an array of points in a path, with the substructure explained below) by filling in the control points. The boolean `cyclic` is used to determine whether the path should be the equivalent of `--cycle`. If the return value is `false`, there is an extra return argument containing the error string.
On entry, the individual knot tables can contain the six knot field values mentioned above (but typically the `left_{x,y}` and `right_{x,y}` will be missing). `{x,y}_coord` are both required. Also, some extra values are allowed:

`left_tension`	number	A tension specifier
`right_tension`	number	like `left_tension`
`left_curl`	number	A curl specifier
`right_curl`	number	like `left_curl`
`direction_x`	number	x displacement of a direction specifier
`direction_y`	number	y displacement of a direction specifier

Issues to watch out for

All the 'normal' requirements for MetaPost paths still apply using this new interface. In particular

- A knot has either a direction specifier, or a curl specifier, or a tension specification, or explicit control points, with the additional note that tensions, curls and control points are split in a left and a right side (directions apply to both sides equally).
- The absolute value of a tension specifier should be more than 0.75 and less than 4096.0, with negative values indicating '`atleast`'.
- The absolute value of a direction or curl should be less than 4096.0.
- If a tension, curl, or direction is specified, any existing control points will be replaced by the newly computed value.

Parsing PDF content streams with LuaTEX

Taco Hoekwater

The new pdfparser library in LuaTEX allows parsing of external pdf content streams directly from within a LuaTEX document. This paper explains its origin and usage.

Background

Docwolves main product is an infrastructure to facilitate paperless meetings. One part of the functionality is handling meeting documents, and to do so it offers the meeting participants a method to distribute, share, and comment on such documents by means of an intranet application as well as an iPad App.

Meeting documents typically consist of a meeting agenda, followed by included appendices, combined into a single pdf file. Such documents can have various revisions, for example if a change has been made to the agenda or if an appendix has to be added or removed. After such a change, a newly combined pdf document is re-distributed.

Annotations can be made on these documents and these can then be shared with other meeting participants, or just communicated to the server for save keeping. Like documents, annotations can be updated as well.

All annotations are made on the iPad, with an (implied) author and an intended audience. Annotations apply to a specific part of the source text, and come in a few types (highlight, sticky note, freehand drawing). The iPad App communicates with a network server to synchronize shared annotations between meeting participants.

The annotation update problem

The server-client protocol aims to be as efficient as possible, especially in the case of communication with the iPad app, since bandwidth and connection costs can be an issue.

This means that for any annotation on a referenced document, only the document's internal identification, the (pdf) page number, and the beginning and end word indices on the page and are communicated back and forth. This is quite efficient, but gives rise to the following problem:

> When a document changes, e.g. if an extra meeting item is added, all annotations following that new item have to be updated because their placement is off.

The actual update process is quite complicated, but the issue this paper deals with is that the server software needs to know what words are on any pdf page, as well as their location on that page, and therefore its text extraction process has to agree with the same process on the iPad.

pdf text extraction

Text extraction is a two-step process. The actual drawing of a pdf page is handled by PostScript-style postfix operators. These are contained in objects that are called page content streams.

After decompression of the pdf, the beginning of a content stream could look like this:

```
59 0 obj
<< /Length 4013 >>
stream
0 g 0 G
1 g 1 G
q
0 0 597.7584 448.3188 re f
Q
0 g 0 G
1 0 0 1 54.7979 44.8344 cm
...
```

Here g, G, q, re, f, Q, and cm are all (postfix) operators, and the numeric quantities are all arguments. As you see, not all operators take the same amount of arguments (g takes one, q zero, and re four). Other operators may take for instance string-valued arguments instead of numeric ones. There are a little over half a dozen different types.
To process such a stream easily, it is best to separate the task (at least conceptually) into two separate tasks. First there is a the lexing stage, which entails converting the actual bytes into combinations of values and types (tokens) that can be acted upon.
Separate from that, there is the interpretation stage, where the operators are actually executed with the tokenized arguments that have preceded it.

pdf text extraction on the iPad

It is very easy on an iPad to display a representation of a pdf page, and Apple also provides a convenient interface to do the actual lexing of pdf content streams that is the first step in getting the text from the page. But to find out where the actual pdf objects are, one has to interpret the pdf document stream oneself, and that is the harder part of the text extraction operation.

pdf text extraction on the server

On the server side, there is a similar problem at a different stage: displaying a pdf is easy, and even literal text extraction is easy (with tools like pdftotext). However, that does not give you the location of the text on the page. On the server, Apple's lexing interface is not available, and the available pdf library (libpoppler) does not offer similar functionality.

Our solution

We needed to write text extraction software that can be used on both platforms, to ensure that the same releases of server and iPad software always agreed perfectly on the what and where of the text on the pdf page.
Both platforms use a stream interpreter written by ourselves in C, with the iPad sofware starting from the Apple lexer, and the server software starting from a new lexer written from scratch.
The prototype and first version of the newly created stream interpreter as well as the server-side lexer were written in Lua. LuaTEX's epdf libpoppler bindings to Lua were a very handy tool at that stage (see below). The code was later converted back to C for compilation into a server-side helper application as well as the iPad App, but originally it was written als a TEXLua script.
A side effect of this development process is that the lexer could be offered as a new LuaTEX extension, and that is exactly what we did.

About the 'epdf' library

This library is written by Hartmut Henkel, and it provides Lua access to the poppler library included in LuaTEX. For instance, it is used by ConTEXt for keeping links in external pdf figures. The library is fairly extensive, but a bit low-level, because it closely mimics the libpoppler interface. It is fully documented in the LuaTEX reference manual, but here is a small example that extracts the page cropbox information from a pdf document:

```
local function run (filename)
   local doc = epdf.open(filename)
   local cat = doc:getCatalog()
   local numpages = doc:getNumPages()
   local pagenum  = 1
   print ('Pages: ' .. numpages)
   while pagenum <= numpages do
      local page = cat:getPage(pagenum)
      local cbox = page:getCropBox()
      print (string.format(
           'Page %d: [%g %g %g %g]',
           pagenum, cbox.x1, cbox.y1,
           cbox.x2, cbox.y2))
      pagenum = pagenum + 1
   end
end
run(arg[1])
```

Lexing via poppler

As said, a lexer converts bytes in the input text stream into tokens, and such tokens have types and values. `libpoppler` provides a way to get one byte from a stream using the `getChar()` method, and it also applies any stream filters beforehand, but it does not create actual tokens.

Poppler limitations

There is no way to get the full text of a stream immediately, it has to be read byte by byte.
Also, if the page content consists of an array of content streams instead of a single entry, the separate content streams have to be manually concatenated.
And content streams have to be 'reset' before the first use.
Here is a bit of example code for reading a stream, using the `epdf` library:

```
function parsestream(stream)
  local self = { streams = {} }
  local thetype = type(stream)
  if thetype == 'userdata' then
    self.stream = stream:getStream()
  elseif thetype == 'table' then
    for i,v in ipairs(stream) do
      self.streams[i] = v:getStream()
    end
    self.stream = table.remove(
                        self.streams,1)
  end
  self.stream:reset()
  local byte = getChar(self)
  while byte >= 0 do
    ...
    byte = getChar(self)
  end
  if self.stream then
    self.stream:close()
  end
end
```

In the code above, any interesting things you want to happen have to inserted at the `...` line.

The example makes use of one helper function (`getChar`) and that looks like this:

```
local function getChar(self)
  local i = self.stream:getChar()
  if (i<0) and (#self.streams>0) then
    self.stream:close()
    self.stream = table.remove(
                      self.streams, 1)
    self.stream:reset()
    i = getChar(self)
  end
  return i
end
```

Our own lexer: 'pdfscanner'

The new lexer we wrote does create actual tokens. Its Lua interface accepts either a poppler stream, or an array of such streams. It puts pdf operands on an internal stack and then executes user-selected operators.
The library `pdfscanner` has only one function, `scan()`. Usage looks like this:

```
require 'pdfscanner'
function scanPage(page)
  local stream = page:getContents()
  local ops = createOperatorTable()
  local info = createParserState()
  if stream then
    if stream:isStream()
       or stream:isArray() then
      pdfscanner.scan(stream, ops,
                       info)
    end
  end
end
```

The functions `createOperatorTable()` and `createParserState()` are helper functions that create arguments of the proper types.

The `scan()` function

As you can see, the function takes three arguments:
The first argument should be either a pdf stream object, or a pdf array of pdf stream objects (those options comprise the possible return values of `<Page>:getContents()` and `<Object>:getStream()` in the epdf library).
The second argument should be a Lua table where the keys are pdf operator name strings and the values are Lua functions (defined by you) that are used to process those operators. The functions are called whenever the scanner finds one of these pdf operators in the content stream(s).
Here is a possible definition of the helper function `createOperatorTable()`:

```
function createOperatorTable()
  local ops = {}
  -- handlecm is listed below
  ops['cm'] = handlecm
  return ops
end
```

The third argument is a Lua variable that is passed on to provide context for the processing functions. This is needed to keep track of the state of the pdf page since pdf operators, and especially those that change the graphics state, can be nested.[1]
In its simplest form, its creation looks like this:

```
function createParserState()
  local stack = {}
  stack[1] = {}
  stack[1].ctm =
    AffineTransformIdentity()
  return stack
end
```

Internally, `pdfscanner.scan()` loops over the input stream content bytes, creating tokens and collecting operands on an internal stack until it finds a pdf operator. If that pdf operator's name exists in the given operator table, then the associated Lua function is executed. After that function has run (or when there is no function to execute) the internal operand stack is cleared in preparation for the next operator, and processing continues.
The processing functions are called with two arguments: the `scanner` object itself, and the `info` table that was passed are the third argument to `pdfscanner.scan`.
The `scanner` argument to the processing functions is needed because it offers various methods to get the actual operands from the internal operand stack.

Extracting tokens from the scanner

The most low-level function in `scanner` is `scanner:pop()` which pops the top operand of the internal stack, and returns a lua table where the object at index one is a string representing the type of the operand, and object two is its value.
The list of possible operand types and associated lua value types is:

`integer`	<number>
`real`	<number>
`boolean`	<boolean>
`name`	<string>
`operator`	<string>
`string`	<string>
`array`	<table>
`dict`	<table>

In case of `integer` or `real`, the value is always a Lua (floating point) number.
In case of `name`, the leading slash is always stripped.
In case of `string`, please bear in mind that pdf actually supports different types of strings (with different encodings) in different parts of

[1] In Lua this could actually have been handled by upvalues or global variables. The third argument was initially a concession made to the planned conversion to C.

the pdf document, so you may need to reencode some of the results; pdfscanner always outputs the byte stream without reencoding anything. pdfscanner does not differentiate between literal strings and hexidecimal strings (the hexadecimal values are decoded), and it treats the stream data for inline images as a string that is the single operand for EI.
In case of array, the table content is a list of pop return values.
In case of dict, the table keys are pdf name strings and the values are pop return values.

While parsing a pdf document that is known to be valid, one usually knows beforehand what the types of the arguments will be. For that reason, there are few more scanner methods defined:

- popNumber() takes a number object off the operand stack.
- popString() takes a string object off the operand stack.
- popName() takes a name object off the operand stack.
- popArray() takes an array object off the operand stack.
- popDict() takes a dictionary object off the operand stack.
- popBool() takes a boolean object off the operand stack.

A simple operator function could therefore look like this (The Affine... functions are left as an exercise to the reader):

```
function handlecm (scanner, info)
  local ty = scanner:popNumber()
  local tx = scanner:popNumber()
  local d  = scanner:popNumber()
  local c  = scanner:popNumber()
  local b  = scanner:popNumber()
  local a  = scanner:popNumber()
  local t =
    AffineTransformMake(a,b,c,d,tx,ty)
  local stack = info.stack
  local state = stack[#stack]
  state.ctm =
    AffineTransformConcat(state.ctm,t)
end
```

Finally, there is also the scanner:done() function which allows you to abort processing of a stream once you have learned everything you want to learn. This comes in handy while parsing /ToUnicode, because there usually is trailing garbage that you are not interested in. Without done, processing only ends at the end of the stream, wasting CPU cycles.

Summary

The new pdfparser library in LuaTEX allows parsing of external pdf content streams directly from within a LuaTEX document. While this paper explained its usage, the formal documentation of the new library is the LuaTEX reference manual. Happy LuaTEX-ing!

MFLua: Instrumentation of MF with Lua

Luigi Scarso

We present MFLua, a MetaFont version which is capable of code instrumentation and has an embedded Lua interpreter that allows glyphs curves extraction and post-processing. We also show and discuss an example of a MetaFont source processed by MFLua to output an OpenType font.

1. Introduction

MFLua is a version of MetaFont, Knuth's program (Knuth, 1986b) designed to draw fonts. MFLua has an embedded Lua interpreter, as well as the capability of the Pascal-WEB code instrumentation to output information about bitmaps and curves used in glyphs drawing. The latest capability is known as *code tracing*. MFLua's main goal is to ease the production of vector fonts which source code is a MetaFont code. MFLua doesn't extend the MetaFont language in any way (i.e., it's not possible to embed Lua code in a MetaFont source file), so that a MetaFont source file is fully compatible with MFLua and vice versa. MFLua won't be extended like LuaTEX extends pdfLATEX. The code instrumentation is a facility to gather and manage information collected in the log file when MetaFont `tracing` instructions are enabled. MFLua automatically saves data into Lua tables using external Lua scripts. Therefore a programmer can manage these tables according to his needs, i.e. extracting a glyph vector outline(s). Rephrasing the previous statements, MFLua is a (bitmap) *tracing* program that knows curves in advance instead of determining them from the bitmap. Please notice that this is only possible when the data have been gathered.
The paper has the following structure: after explaining what *code instrumentation* is (section 8.2.), it shows the components used to trace a glyph (section 8.3.) and finally two different approaches to manage curves (section 8.4.).
As a final remark, we consider MFLua as being in a state between a proof of concept and alpha and it's not (yet) too user-friendly. Its code is hosted at https://github.com /luigiScarso/mflua.

2. Code instrumentation

MetaFont is written in Pascal-WEB (a programming language by Donald Knuth to have a real literate programming tool. As the name suggests, it's a subset of Pascal) and is automatically translated into C by `tangle` and `web2c`. Instrumentation is the capability to add *trace statements* (a.k.a. *sensors*) in strategic points of the code to register current state information and pass it to the Lua interpreter. A typical sensor has the `mflua` prefix. We can see some sensors in the following chunk of code, the main body of MetaFont (slightly refolded to fit the printing area).

```
@p begin @!{|start_here|}
mflua_begin_program;
{in case we quit during initialization}
history:=fatal_error_stop;
t_open_out; {open the terminal for output}
if ready_already=314159 then goto start_of_MF;
@<Check the ``constant'' values...@>@;
if bad>0 then
  begin wterm_ln('Ouch---my internal constants
     have been clobbered!',
    '---case ',bad:1);
@.Ouch...clobbered@>
  goto final_end;
  end;
{set global variables to their starting values}
initialize;
@!init if not get_strings_started then
  goto final_end;
init_tab; {initialize the tables}
init_prim; {call |primitive| for each primitive}
init_str_ptr:=str_ptr; init_pool_ptr:=pool_ptr;@/
max_str_ptr:=str_ptr; max_pool_ptr:=pool_ptr;
fix_date_and_time;
tini@/
ready_already:=314159;
mfluaPRE_start_of_MF;
start_of_MF: @<Initialize the output routines@>;
@<Get the first line of input and prepare
  to start@>;
history:=spotless; {ready to go!}
mflua_initialize;
if start_sym>0 then
  {insert the `\&{everyjob}' symbol}
  begin cur_sym:=start_sym; back_input;
  end;
mfluaPRE_main_control;
main_control; {come to life}
mfluaPOST_main_control;
final_cleanup; {prepare for death}
mfluaPOST_final_cleanup;
end_of_MF: close_files_and_terminate;
final_end: ready_already:=0;
end.
```

We're going to examine the role of the `mflua_begin_program` sensor. The Pascal-into-C translator, web2c, is smart enough to distinguish a symbol already present in the Pascal source from an external symbol (i.e., a symbol defined in another file). In the latter case, the programmer has to register that symbol into the file `texmf.defines` if the symbol is related to an argumented procedure: the translator will manage properly the arguments translation. The translated code will contain the C form of the sensor symbol, which will be resolved at compile-time — i.e., we need an object file that contains that symbol. Each sensor is stored into `mflua.h` and `mflua.c`. The first one lists the symbol:

```
#include "lua51/lua.h"
#include "lua51/lualib.h"
#include "lua51/lauxlib.h"
#include <kpathsea/c-proto.h>
#include <web2c/config.h>

extern lua_State* Luas[];
extern int mfluabeginprogram();
```

while the second one contains the corresponding function source code:

```
int mfluabeginprogram()
{
  lua_State *L = luaL_newstate();
  luaL_openlibs(L);
  Luas[0] = L;
   /* execute Lua external "begin_program.lua" */
  const char* file = "begin_program.lua";
  int res = luaL_loadfile(L, file);
  if ( res==0 ) {
      res = lua_pcall(L, 0, 0, 0);
    }
  priv_lua_reporterrors(L, res);
  return 0;
}
```

The above function initializes the Lua interpreter, stores its state in the array `Luas[]` (it would be possible to have more than one interpreter but this feature is currently neglected) and then executes the external script `begin_program.lua` calling `lua_pcall(L, 0, 0, 0)`. This call protects the interpreter from errors. Every time we run `mf` (the MetaFont program), it loads and executes the Lua script `begin_program.lua`, customizable by programmers.
We surely need to pay attention to some issues. The first one is that literate programming style allows to collect every changes we make in a source file into a *change* file (`mf.ch` in our case), which is then merged into a Pascal program by `tangle`. This means that inserting a sensor can interfere with the change file. In this case we also have to insert the sensor into the change file as we do, for instance, with `mfluaPRE_make_ellipse(major_axis, minor_axis,theta,tx,ty,0)`. Of course the right solution is directly inserting the sensors in the change file. Unfortunately it's usually faster discovering where to insert a sensor in the source file and then managing conflicts in the change file: source files have a context — the source itself — that change file don't. The second issue is the need to export some MetaFont variables and constants to the Lua interpreter. An easy way to accomplish this task is inspecting the translated C code to realize how those variables and constants are managed (e.g., to make Lua read the `charcode` variable, which contains the index of the current glyph, we need to know that it's stored into the `internal` array at index 18 (the index is from the MetaFont WEB source) so that we can write a wrapper function like the following one:

```
#define roundunscaled(i) (((i>>15)+1)>>1)
EXTERN scaled internal[maxinternal + 1]  ;
static int
 priv_mfweb_LUAGLOBALGET_char_code(lua_State *L)
{
  integer char_code=18;
  integer p=roundunscaled(internal[char_code])%256;
  lua_pushnumber(L,p);
  return 1;
}
```

and then make it available to the Lua interpreter as the LUAGLOBALGET_char_code variable:

```
int mfluainitialize()
{
  /* execute Lua external "mfluaini.lua" */
  lua_State *L = Luas[0];
  /* register lua functions */
:
lua_pushcfunction(L,
       priv_mfweb_LUAGLOBALGET_char_code);
lua_setglobal(L,"LUAGLOBALGET_char_code");
:

/* execute Lua external "mfluaini.lua" */
  const char* file = "mfluaini.lua";
  int res = luaL_loadfile(L, file);
  if ( res==0 ) {
      res = lua_pcall(L, 0, 0, 0);
  }
  priv_lua_reporterrors(L, res);
  return 0;
}
```

Users can read and set this variable though the set value won't be passed to MetaFont in order to interfere as little as possible with its state. That's why we prefer to inspect the translated C code, which quality depends on the translation performed at compile-time. A clean solution should only depend on the WEB source. For historical reasons, translating code from Pascal into C outputs two files (mf0.c and mf1.c). Finding a symbol implies searching in two files, which hardens the process.
There are currently 24 sensors, 33 global variables and 15 scripts, though it's possible to increase these quantities if we discover that tracing a specific function inside MetaFont is better than reimplementing it in Lua. While it's easy to implement the algorithm to draw a curve in Lua, it's slightly harder to implement the algorithm to fill a region. Whatever, the main goal is to keep the number of sensors as low as possible. Notice that MFLua currently reads scripts from the current folder and doesn't use the standard TEX folders.

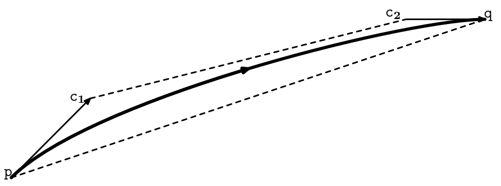


Figure 1: A cubic Bézier curve and its convex hull.

The counterpart of `mflua_begin_program` is `mflua_end_program`, which calls the `end_program.lua` script. It contains all the functions used to transform the components of a glyph, the subject of the next section.

3. The components of a glyph

MetaFont mainly manages Bézier cubic curves (see fig. 1).[1] This curve is completely described by four *controls points*: $\mathbf{p}$ (called the *starting point*), $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{q}$ (known as the *ending point*). The MetaFont command to draw such a curve is

```
draw p .. controls c₁ and c₂ .. q;
```

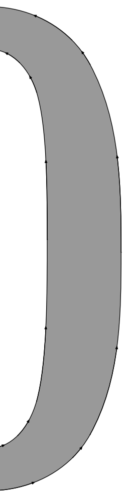
This curve lies on the XY plane and its *parametric form* is quite simple:

$$\mathbf{B(t) = (1-t)^3p + 3(1-t)^2tc_1 + 3(1-t)t^2c_3 + t^3q} \qquad \forall \mathbf{t} \in [\mathbf{0},\mathbf{1}] \tag{1}$$

The corresponding algebraic expression, the *closed form*, is more complex but it's useful to quickly test whether a point belongs to the curve or not.
Equation 1 has first derivatives $\mathbf{B'(0) = 3(c_1 - p)}$ and $\mathbf{B'(1) = 3(q - c_2)}$ respectively when $t = 0$ and $t = 1$. We can easily calculate them when we know $\mathbf{p}$, $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{q}$. An important property is that a Bézier cubic curve is completely contained in the polygon $\mathbf{p\,c_1\,c_2\,q\,p}$ (the convex hull) and this immediately leads to the conclusion that the intersection of two curves is empty if and only if the intersection of their convex hulls is empty. Another important property is the existence of the *De Casteljau's algorithm*, very easy to implement: given the four control points of a curve and a time t_1, it returns the point $(x_1, y_1) = \mathbf{B(t_1)}$ on the curve and the two series of control points, one for the curve $\mathbf{B_l(t) = B(t), t} \in [\mathbf{0}, \mathbf{t_1}]$ (the *left side*) and one for the curve $\mathbf{B_r(t) = B(t), t} \in [\mathbf{t_1}, \mathbf{1}]$ (the *right side*). It recursively reduces the curve $\mathbf{B(t)}$, $\mathbf{t} \in [\mathbf{0}, \mathbf{1}]$ tracing to the tracing of the left side $\mathbf{B_l(t) = B(t)}$, $\mathbf{t} \in [\mathbf{0}, \mathbf{1/2}]$ and the right side $\mathbf{B_r(t) = B(t)}$, $\mathbf{t} \in [\mathbf{1/2}, \mathbf{1}]$ (the recursion ends when the distance between two points (x_j, y_j) and (x_{j+1}, y_{j+1}) is less than a pixel).
The De Casteljau's algorithm is useful because it estimates how long a curve is counting the number of pixels covered by the curve. It also finds intersections of two curves $\mathbf{B(t)}$ and $\mathbf{C(t)}$ reducing this problem to the problem of calculating the intersection of four curves: left and right side of $\mathbf{B(t)}$ and left and right side of $\mathbf{C(t)}$ for $t = 1/2$. The algorithm keeps working when one curve degenerates into a segment (i.e., if $\mathbf{p = c_1}$ and $\mathbf{c_2 = q}$) or when it degenerates into a point ($\mathbf{p = c_1 = c_2 = q}$). Therefore it can be used to find an intersection between a line and a curve and to test if a point belongs to a curve. A set of curves $\{\mathbf{B_1}, \mathbf{B_2} \ldots \mathbf{B_n}\}$ where $\mathbf{q_{j-1} = p_j}$ and $\mathbf{q_n = p_0}$ is a simple cycle

[1] I borrow notation from Marsh (2005), where points and functions in the Bézier curves section are represented by bold, upright letters.

if the only intersection is $(x, y) = \mathbf{q_n} = \mathbf{p_0}$. Simple cycles are the building blocks of a glyph: a simple cycle can be filled or unfilled and, according to MetaFont's point of view, a glyph is a set of cycles filled and/or unfilled at the right moment.
A normal MetaFont designer doesn't care about these details because MetaFont has a high level language to describe curves, points, lines, intersections, filled and unfilled cycles and, most important, pens. The listed entities produce a combination of two different basic draws: regions (un)filled by a *contour* and regions (un)filled by the stroke of a pen, i.e., the *envelope of a pen*. Both are simple cycles, but their origin is very different.
Let's consider the code of the glyph 0 from the file `xmssdc10.mf`:


of the numeral 0
t.

```
 cmchar "The numeral 0";
beginchar("0",9u#,fig_height#,0);
italcorr fig_height#*slant-.5u#;
adjust_fit(0,0);
penpos1(vair,90);
penpos3(vair,-90);
penpos2(curve,180);
penpos4(curve,0);
if not monospace:
 interim superness:=sqrt(more_super*hein_super);
fi
x2r=hround max(.7u,1.45u-.5curve);
x4r=w-x2r; x1=x3=.5w;
y1r=h+o; y3r=-o;
y2=y4=.5h-vair_corr;
y2l:=y4l:=.52h;
penstroke pulled_arc.e(1,2) & pulled_arc.e(2,3)
 & pulled_arc.e(3,4)
 & pulled_arc.e(4,1) & cycle;  % bowl
penlabels(1,2,3,4);
endchar;
```

Fig. 2 shows a glyph only made by two contours which are the result of `penpos` and `penstroke` macros. Of course we could obtain the same result drawing 24 curve sections (12 for the outer contour, 12 for the inner one) but it should be clear that the MetaFont description is much more straight or, at least, 'typographic'.
Things completely change when we consider the numeral 2:

```
cmchar "The numeral 2";
beginchar("2",9u#,fig_height#,0);
italcorr fig_height#*slant-.5u#;
adjust_fit(0,0);
numeric arm_thickness, hair_vair;
hair_vair=.25[vair,hair];
arm_thickness= Vround(if hefty:slab+2stem_corr else:.4[stem,cap_stem] fi);
pickup crisp.nib;
pos7(arm_thickness,-90); pos8(hair,0);
bot y7r=0; lft x7=hround .9u; rt x8r=hround(w-.9u);
y8=good.y(y7l+beak/2)+eps;
arm(7,8,a,.3beak_darkness,beak_jut);%arm and beak
```

```
pickup fine.nib; pos2(slab,90);
pos3(.4[curve,cap_curve],0);
top y2r=h+o; x2=.5(w-.5u);
rt x3r=hround(w-.9u); y3+.5vair=.75h;
if serifs:
 numeric bulb_diam;
 bulb_diam=hround(flare+2/3(cap_stem-stem));
 pos0(bulb_diam,180); pos1(cap_hair,180);
 lft x1r=hround .9u; y1-.5bulb_diam=2/3h;
 (x,y2l)=whatever[z1l,z2r];
 x2l:=x; bulb(2,1,0);  % bulb and arc
else: x2l:=x2l-.25u; pos1(flare,angle(-9u,h));
 lft x1r=hround .75u; bot y1l=vround .7h;
 y1r:=good.y y1r+eps; x1l:=good.x x1l;
 filldraw stroke term.e(2,1,left,.9,4);
fi  % terminal and arc
pos4(.25[hair_vair,cap_stem],0);
pos5(hair_vair,0);
pos6(hair_vair,0);
y5=arm_thickness; y4=.3[y5,y3];
top y6=min(y5,slab,top y7l);
lft x6l=crisp.lft x7;
z4l=whatever[z6l,(x3l,bot .58h)];
z5l=whatever[z6l,z4l];
erase fill z4l--
 z6l--lft z6l--
 (lft x6l,y4l)--cycle;%erase excess at left
filldraw stroke z2e{right}..tension
  atleast .9 and atleast 1
 ..z3e{down}..{z5e-z4e}z4e--z5e--z6e;%stroke
penlabels(0,1,2,3,4,5,6,7,8);
endchar;
```

As we can see in fig. 3, there are both a contour and envelopes of more than a pen; there are intersections between the contour the envelopes and the pens, and some curves are outside the glyph (some of these curves are used to delete unwanted black pixels). There are also some unexpected straight lines and small curves. The number of curves looks quite large, which is not what we desire as we want to obtain the outline depicted in fig. 4.
Unfortunately, things are even different and it's necessary to describe how MetaFont calculates pen envelopes to go on. This is explained in the book 'MetaFont: The Program' (Knuth, 1986a) at the 'Polygonal pens' part, chapter 469, that we briefly quote with a slightly modified notation:

"Given a convex polygon with vertices $\mathbf{w_0}, \mathbf{w_1}, \ldots, \mathbf{w_{n-1}}, \mathbf{w_n} = \mathbf{w_0}$ a in *counterclockwise* order …(and a curve $\mathbf{B}(\mathbf{t})$) the envelope is obtained if we offset $\mathbf{B}(\mathbf{t})$ by $\mathbf{w_k}$ when the curve is travelling in a direction $\mathbf{B}'(\mathbf{t})$ lying between the directions $\mathbf{w_k} - \mathbf{w_{k-1}}$ and $\mathbf{w_{k+1}} - \mathbf{w_k}$. At times t when the curve direction $\mathbf{B}'(\mathbf{t})$ increases past $\mathbf{w_{k+1}} - \mathbf{w_k}$, we temporarily stop plotting the offset curve and we insert a straight line from $\mathbf{B}(\mathbf{t}) + \mathbf{w_k}$ to $\mathbf{B}(\mathbf{t}) + \mathbf{w_{k+1}}$; notice that this straight line is tangent to the to the offset curve. Similarly, when the curve direction decreases past $\mathbf{w_k} - \mathbf{w_{k-1}}$, we stop plotting and insert a straight line from $\mathbf{B}(\mathbf{t}) + \mathbf{w_k}$ to $\mathbf{B}(\mathbf{t}) + \mathbf{w_{k+1}}$; the latter line is actually a 'retrograde' step which will not be part of the final envelope under the MetaFont's assumptions. The result of this construction is a continuous path that consist of alternating curves and straight line segments."

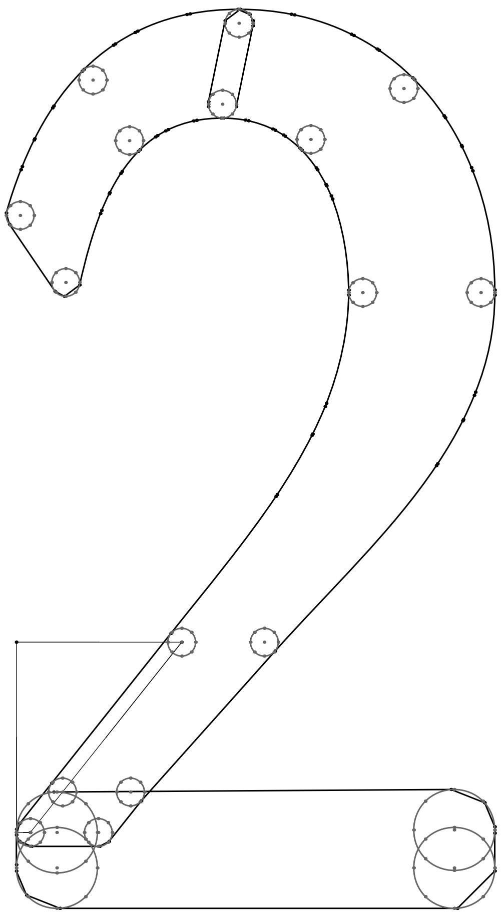
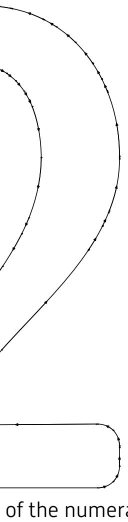


Figure 3: The glyph of the numeral 2 in `xmssdc10` font. We can see envelopes and pens (thick curves) and a contour (thin curve).



of the numeral 2
t.

This explains why the number of the curves is large and why there are small curves, but says nothing about those circular curves that we can see in fig. 4: MetaFont indeed converts an elliptical pen into a polygonal one and then applies the algorithm. The conversion is accurate enough to guarantee that the envelope is correctly filled with the right pixels. This is a key point to understand: *MetaFont's main task is to produce the best bitmap of a glyph, not the best outline*.
The role of the sensors is to gather as much information as possible about pixels, contours, the polygonal version of the pens, envelopes and their straight lines and then store these information (basically the edge structure of the pixels and Bézier curves with an eventual offset) into appropriate Lua tables. As MetaFont halts, the Lua interpreter calls `end_program.lua` and let the programmer manage these tables: sometimes, as we have seen in the numeral 0 case, the post-process can be quite simple, sometimes not. MFLua doesn't automatically output a glyph outline because it's the programmer who has to implement the best strategy according to his experience.

4. Two different strategies for post-processing the curves

4.1 The Concrete Roman 10 pt

The first use of MFLua has been the post-processing of Concrete Roman 10 pt to obtain an OpenType version of it. This font is described in the file `ccr10.mf`. As we previously said, sensors collect the data into Lua tables and `end_program.lua` post-processes them at the end of the execution (we could even choose to execute the no-more post-process during the execution). The script `end_program.lua` defines the global array `chartable[index]` that contains the data for the glyph with char code `index`: we have the edge structure that allows the program to calculate the pixels of the glyph as well as the three arrays `valid_curves_c`, `valid_curves_e` and `valid_curves_p` that gather the data of contours, envelopes and the polygonal version of the pens. Each array contains the array of the control points `{p,c1,c2,q}` stored as a string `"(<x>,<y>)"`, where <x> and <y> are the coordinates of the point. With fig. 3 as a reference, we can see that when we draw a glyph with a pen it usually has overlapping strokes. Along with the curves of the pen(s), these overlaps create curves inside or outside the glyph that must be deleted. Having the pixels of the glyph, we can use the parametric form 1 to check if a point (x, y) (or better, a neighborhood with center (x, y)) is inside or outside. If all the points of the curve are inside or outside, we can delete them. The drawback is that while time t goes linearly in $\mathbf{B}(\mathbf{t})$, the points $(x(t), y(t))$ follow a cubic (i.e., not linear) law in case the curve is not a straight line. Hence, they are not equally spaced — this means that we can jump over some critical points. Using the same time interval steps for each curve means that short curves are evaluated in times where the points can differ less than a pixel — a useless evaluation. Of course not all the curves are inside the glyph: there are curves on the border and curves partially lying on the border and partially inside (or outside). In the latter case the result of evaluation is an array of time intervals where the curve crosses the border.
Once we have deleted the curves that are completely inside (or outside) the glyph, the next step is to merge all the curves and split them using the previously seen time interval (this is done by a Lua implementation of the De Casteljau's algorithm). Now we have a set of curves that are on the border or 'near' it (i.e., partially on the border). We can delete those curves having only one intersection (a *pending curve*), supposing that each curve of the final outline has exactly 2 intersections at times $t_0 = 0$ and $t_1 = 1$.
To calculate all the intersections we use the following trick: if we have n curves, we produce a MetaFont file that contains the code that calculates the intersection between p_i and p_j for $1 \leq j \leq n$ and $j < i \leq n$ (given that $p_j \cap p_i = p_i \cap p_j$) and then we parse the log file with Lua. For example if

```
p1={"(57.401,351.877)", "(57.401,351.877)",
    "(57.901,349.877)", "(57.901,349.877)"}
```

and

```
p2={"(56.834,356.5)",   "(56.834,354.905)",
    "(57.031,353.356)", "(57.401,351.877)"}
```

then we have

```
batchmode;
message "BEGIN i=2,j=1";
path p[];
p1:=(57.401,351.877) ..
  controls (57.401,351.877) and (57.901,349.877) ..
   (57.901,349.877);
p2:=(56.834,356.5) ..
  controls (56.834,354.905) and (57.031,353.356) ..
    (57.401,351.877);
numeric t,u;
(t,u) = p1 intersectiontimes p2;
show t,u;
message "" ;
```

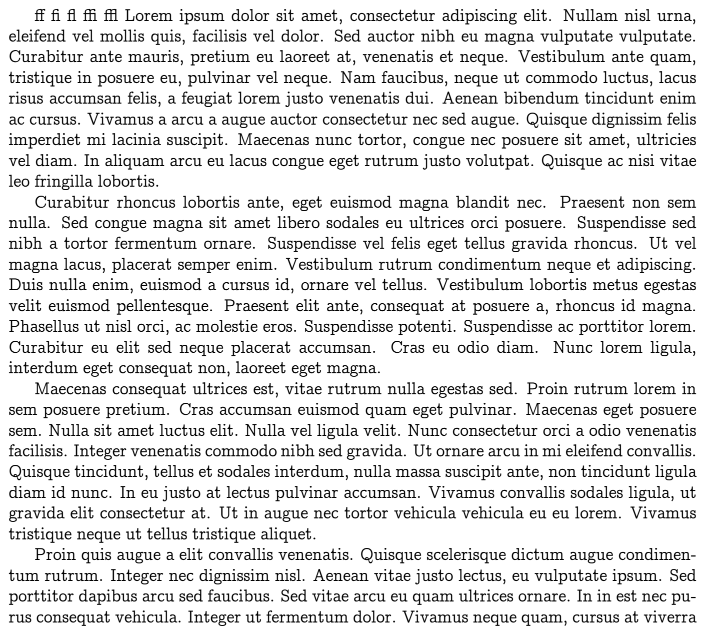
and the log

```
BEGIN i=2,j=1
>> 0
>> 0.99998
```

If the result is $(-1,-1)$ the intersection is empty. There are two problems with this approach: the first one shows when a curve crosses the border and time intervals can generate two curves, one completely outside and one completely inside — hence deleting an intersection. To avoid this issue we must adjust the intervals moving the extremes a bit. We have the second problem when there can be curves with three or more intersections — i.e., we can have *loops*. Opening a loop can be a difficult task: e.g., if the curve p_a intersects $\{p_b, p_c, p_d\}$ at the same time t_a and p_b intersects $\{p_a, p_c, p_d\}$ at t_b then $I_a = \{p_a\} \cup \{p_b, p_c, p_d\}$ is equal to $I_b = \{p_b\} \cup \{p_a, p_c, p_d\}$ and we can delete p_a and p_b because p_c and p_d stay connected. But with more than three intersections things become more complex.
To solve these cases, `end_program.lua` has a series of *filters*. A filter acts on a specific glyph and typically removes unwanted curves and/or adjusts the control points to ensure that a curve joins properly with its predecessors and successor. Of course this means that the programmer inspects each glyph separately, which is reasonable when we are designing the font — less reasonable when we convert it.
We can call this approach *per-font and per-glyph*: `end_program.lua` is a Lua script valid only for a specific font and which has filters for each glyph.
The script `end_program.lua` also has some functions to convert the outlines (with the correct turning number) of each glyph into a SVG font: this font format can be imported into FontForge and, usually after re-editing the glyphs (tipically simplifying the curves), it can be saved as an OpenType CFF. In fig. 5 we can see an example of this font.

4.2 The Computer Modern Sans Serif Demibold Condensed 10 pt

We now approach a more ‘geometric’ strategy. We don’t want to output an OpenType font but to find an `end_program.lua` more *universal and per-glyph* and less *per-font and per-glyph*. Our experience with `ccr10.mf` make us believe that is always possibile


ﬀ ﬁ ﬂ ﬃ ﬄ Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nisl urna, eleifend vel mollis quis, facilisis vel dolor. Sed auctor nibh eu magna vulputate vulputate. Curabitur ante mauris, pretium eu laoreet at, venenatis et neque. Vestibulum ante quam, tristique in posuere eu, pulvinar vel neque. Nam faucibus, neque ut commodo luctus, lacus risus accumsan felis, a feugiat lorem justo venenatis dui. Aenean bibendum tincidunt enim ac cursus. Vivamus a arcu a augue auctor consectetur nec sed augue. Quisque dignissim felis imperdiet mi lacinia suscipit. Maecenas nunc tortor, congue nec posuere sit amet, ultricies vel diam. In aliquam arcu eu lacus congue eget rutrum justo volutpat. Quisque ac nisi vitae leo fringilla lobortis.

Curabitur rhoncus lobortis ante, eget euismod magna blandit nec. Praesent non sem nulla. Sed congue magna sit amet libero sodales eu ultrices orci posuere. Suspendisse sed nibh a tortor fermentum ornare. Suspendisse vel felis eget tellus gravida rhoncus. Ut vel magna lacus, placerat semper enim. Vestibulum rutrum condimentum neque et adipiscing. Duis nulla enim, euismod a cursus id, ornare vel tellus. Vestibulum lobortis metus egestas velit euismod pellentesque. Praesent elit ante, consequat at posuere a, rhoncus id magna. Phasellus ut nisl orci, ac molestie eros. Suspendisse potenti. Suspendisse ac porttitor lorem. Curabitur eu elit sed neque placerat accumsan. Cras eu odio diam. Nunc lorem ligula, interdum eget consequat non, laoreet eget magna.

Maecenas consequat ultrices est, vitae rutrum nulla egestas sed. Proin rutrum lorem in sem posuere pretium. Cras accumsan euismod quam eget pulvinar. Maecenas eget posuere sem. Nulla sit amet luctus elit. Nulla vel ligula velit. Nunc consectetur orci a odio venenatis facilisis. Integer venenatis commodo nibh sed gravida. Ut ornare arcu in mi eleifend convallis. Quisque tincidunt, tellus et sodales interdum, nulla massa suscipit ante, non tincidunt ligula diam id nunc. In eu justo at lectus pulvinar accumsan. Vivamus convallis sodales ligula, ut gravida elit consectetur at. Ut in augue nec tortor vehicula vehicula eu eu lorem. Vivamus tristique neque ut tellus tristique aliquet.

Proin quis augue a elit convallis venenatis. Quisque scelerisque dictum augue condimentum rutrum. Integer nec dignissim nisl. Aenean vitae justo lectus, eu vulputate ipsum. Sed porttitor dapibus arcu sed faucibus. Sed vitae arcu eu quam ultrices ornare. In in est nec purus consequat vehicula. Integer ut fermentum dolor. Vivamus neque quam, cursus at viverra

Figure 5: The ConcreteOT font produced by MFLua from `ccr10.mf`.

to write a MetaFont program that outputs a nice bitmap of a glyph using a very complex set of curves. This is especially true when we use pens and the need to manually correct every error arises. Up to now we only made few outlines of numerals.

There are new functions to trace a curve and to calculate the intersections between two cubics (both based on De Casteljau's bisection algorithm, an application of De Casteljau's algorithm) so the parametric form and the trick to calculate the intersections are not needed anymore. We also keep contours, envelopes and pens apart almost until the end of the process, when we first merge envelopes and pens and then, at last, contours. The most important enhancement is probably the replacement of the polygonal version of a pen with an elliptical one. MetaFont generates a polygonal getting an ideal ellipse with major axis, minor axis and the angle of rotation from the pen specifications and then calls `make_ellipse`. Putting a sensor around helps us store the axis and theta into a Lua table, to be read later from `end_program.lua`. The next step is a trick again: we call MFLua with the following file:

```
batchmode;
fill fullcircle
    xscaled (majoraxis)
    yscaled (minoraxis)
    rotated (theta) shifted (0,0);
shipit;
bye.
```

where `majoraxis`, `minoraxis` and `theta` get the ellipse data. MFLua then saves the outlines of the filled ellipse into another file, from which they can be read by `end_program.lua`. This script then saves each elliptical pen in a table, with `p..c1..c2..q` as a key to be reused later, instead of the polygonal one. We can see the result in fig. 6: the approximation is quite good. This reduces the total number of curves and gives the glyph a more natural look.

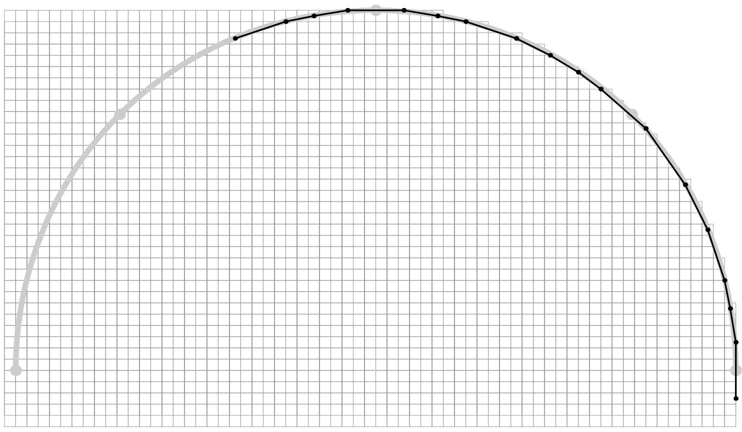


Figure 6: Real polygonal pen (black) vs. calculated elliptical pen (gray). Square boxes are the pixels. It's the bottom right part of fig. 3 .

5. Conclusion

We believe that MFLua is an interesting tool for font designers because too many fonts (if not all) are currently designed using contours. In this case `end_program.lua` should be simple (less or even no intersections, compared to the MetaFont technique, see the numeral 0 of `xmssdc10.mf`). On the other side, using the pens shows that extracting an outline is a difficult task. It's almost impossible to find an always valid script. The outlines from an envelope usually have a large number of curves, which is not a good feature, and this is a MetaFont property: we can always implement routines to simplify them, though FontForge already does it.
The work will continue on `xmssdc10.mf` to find an `end_program.lua` modular and flexible enough for a wide application.

6. References

Knuth, D. E. (1986a). *MetaFont: The Program*. Addison-Wesley, Massachusetts, 1st edition.

Knuth, D. E. (1986b). *The MetaFontbook*. Addison-Wesley, Massachusetts, 1st edition — with final correction made in 1995 —.

Marsh, D. (2005). *Applied Geometry for Computer Graphics and CAD*. Springer, London, 2nd edition.

Conference portfolio
(Workshop)

Willi Egger

In accordance to the conference's theme, a workshop for making a portfolio binder has been held. The portfolio was made so it could carry the papers for the conference, such as preprints of the proceedings, additional papers and the carpenter's pencil given to each participant. The construction is made from a single sheet of cardboard with folded flaps along three sides, so that it completely envelopes the content. The portfolio is held closed by a black elastic band.

1. Introduction

A portfolio is a practical solution to keep all information gathered during a meeting or conference neatly in one place. Most portfolios are made from strong material, for instance manilla cardboard. Normally they are built from several pieces, i.e. the flaps are glued onto the back-cover. In this workshop an intriguing design is used, which allows to prepare the complete portfolio from a single sheet of cardboard without the need for glueing.

2. Basic design

In order to understand the mechanics of this type of construction, it is a good idea to make a blueprint first. Although the same principles apply to the version made with cardboard, it is important to understand that the drawing is only in two dimensions. In other words, it does not include compensation for the thickness of the content and the material used for the protfolio. When making the actual portfolio, we will have to compensate for this.
The size of your portfolio is dictated by it's intended content. Therefor, the height of your blank cardboard sheet should be height (h) + width (w) of the content. The width is calculated by adding twice the width of the content (w) to the width of the fold-in flap (f).
For the real world portfolio, we will take into account the thickness of the content. This adds 2 × the spine width to the width of the sheet.

3. The conference portfolio

In order to determine the size of the portfolio, one first needs to know the dimensions of the content which it should be able to carry.
For the preprints of the proceedings these dimensions are:

- Height = 265 mm
- Width = 210 mm
- Thickness = 9 mm

The space inside the portfolio should always be slightly larger than the actual dimensions of the content, not to mention we have to leave room for our carpenter's pencil.
The final dimensions of the portfolio are set to:

- Height = 275 mm
- Width = 225 mm
- Spine width = 10 mm

When choosing a material for the portfolio, it is important to check that the grain of the material is in the direction **width** axis of the material. The cardboard size should be 500 × 700 mm (width × length).

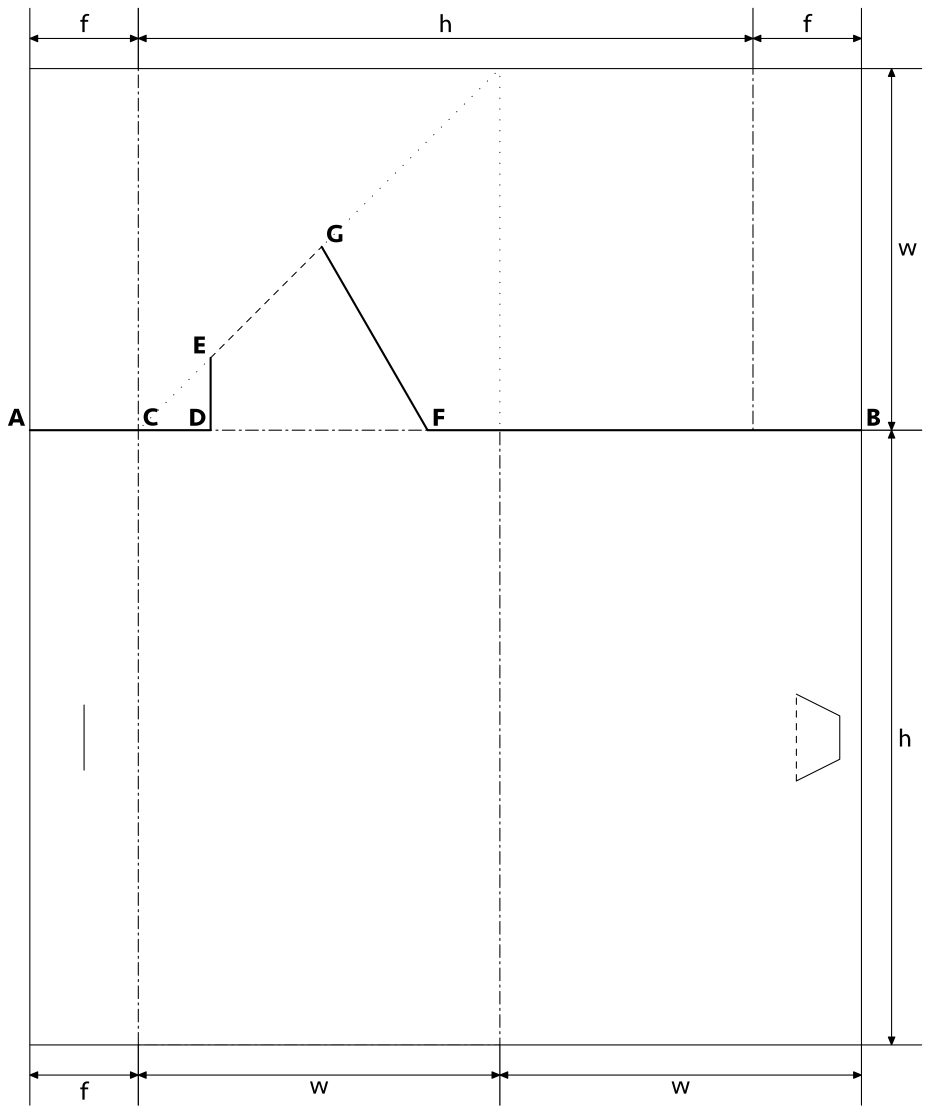


Figure 1: The blueprint for the portfolio

4. Making the portfolio

In order to get good results, make sure to measure and mark precisely ("measure twice, cut once"). For creasing the folds a fairly sharp bone-folder is advised. For cutting, a cutter with snap-off blade is most suitable. The ruler used should be made of steel. Ones made from hard woods such as beechwood are also usable, however one should be careful not to cut into the wood. To make rulers more steady when drawing and cutting, a strip of sandpaper may be glued to the back of them.

When looking at the blueprint discussed earlier, we see we need to compensate for the thickness of the content (in our case, 10mm). In addition to that, we also have to take into account the thickness of the material. The latter is reflected by the fact that point **C** in the drawing 2 is moved to the right from the inner line of the spine by about 2 mm. Thickness of the material affects overall thickness of the portfolio, and also things such as corner radii, as stiffer materials do not fold as easily.

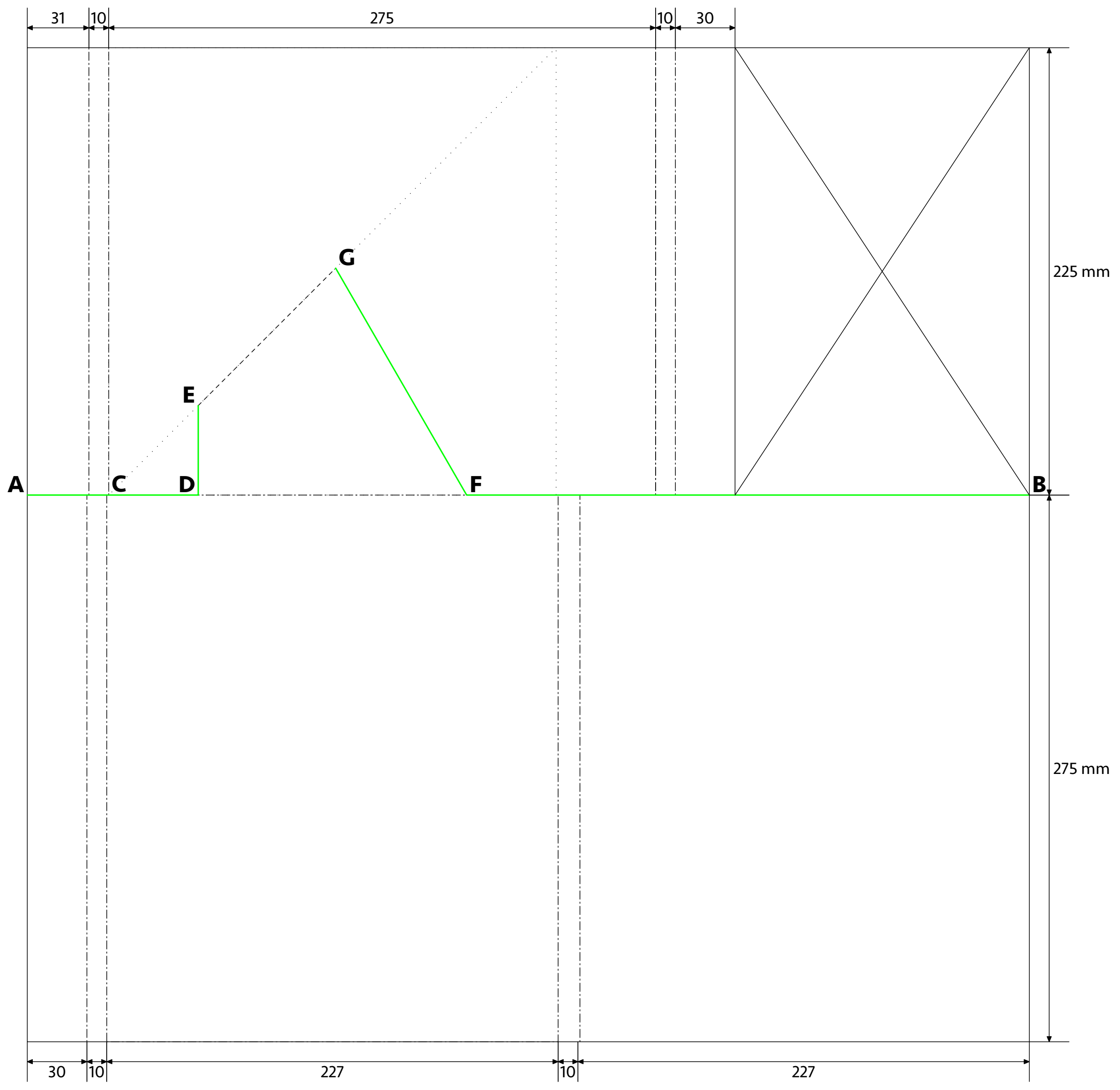


Figure 2: Conference portfolio layout

Correctly creasing the line between points **E** and **G** is crucial. This line must be made at an angle of exactly 45°.
After folding, folds **E – G** and **D – F** should be sharpened with a bone-folder.
For folding narrow spines, it is easiest to put a ruler on the inside along the line you wish to fold. Use a bone-folder to follow the edge of the ruler on the outside. As you drag it along the ruler, you push the material upwards into a straight, clean fold.
The spines are just wide enough to place another crease between the outer spine folds. It is a little more work, but allows the spine to flex if the portfolio contains less paper than the spine width allows for. This guarantees a tight fit when the elastic band is used.
To make the slots used to insert a postcard into the front cover, it is best to prepare a template. The template should be 210 × 160 mm and can be made from a piece of discarded cardboard. Draw a rectangle of 150 × 100 mm, which is offset by 40 mm from the bottom and the right edge. Mark at each angle two points e.g. 15

mm offset along the rectangle's frame. Cut the drawn triangles out of the template or mark the 8 points by punching little holes with a sharp awl. By cutting the triangle a fraction of a millimeter larger than drawn, you can then insert a 150 × 100 mm postcard with ease.
For best results, make your pencil marks as lightly as possible or alternatively use the tip of the bone-folder.
When the portfolio is finished, you could cut the short edges of the fold-in flaps with a bevelled edge.

5. The steps for making the porfolio

- Place the sheet in front of you so the widest side is parallel to the edge of the working table.
- Mark and crease a horizontal line at 275 mm from the bottom of the sheet (A – B).
- Mark and crease a vertical line 30 mm from the left edge extending to the already creased horizontal line (A – B).
- Mark and crease a vertical line 40 mm from the left edge extending up to the already creased horizontal line (A – B).
- Point C is 2 mm right of the last creased line. Draw a line upwards at an angle of 45°starting from C.
- Draw a vertical line 80 mm from the left edge on the horizontally creased line (A – B) (D), until it intersects with the diagonal. This intersection point is point E.
- Draw a line starting 210 mm from the left edge on the horizontally creased line (A – B) (F) upwards until it intersects with the diagonal (the angle is not important) (G).
- Cut lines A – D, D – E, F – B and F – G.
- Crease E – G on the **outside** of the cover. Be very careful when doing this crease, or the portfolio will not fold correctly.
- Fold D – F precisely and sharpen the fold with the bone-folder.
- Fold E – G while turning the strip, fold down after precise positioning of the strip with the bone-folder.
- Mark the fold-in flaps at the bottom and top so that they are approximately 2 mm inside the cover after folding.
  Mark two lines 2 and 12 mm to the right of the fold-in (direction of the height of the portfolio).
- Completely unfold the portfolio.
  Mark and crease a line 10 mm to the left of the bottom turn in. Mark and crease a line 10 mm to the right of the top fold-in.
  Crease also the two lines in direction of the height of the portfolio.
- Cut the top fold-in to the width of the bottom fold-in flap (30 mm).
- There are now four small strips of 10 mm marked by creases. Place another crease between the creased lines if you want to give the portfolio rounded spines.
- All creased lines need to be folded now. Use the ruler and the bone-folder for this task.
- Refold the portfolio and fold the flaps towards the inside.
- Close the portfolio. Mark the width of the cover. Open the cover and check that the marks are at equal distance from the front edge.
- Cut the front cover to size. Alternatively you can also fold the oversized flap toward the inside of the front cover. By glueing the edges it forms a pouch.

Making the closing:
For the closing an elastic band is fixed to the portfolio with two eyelets.

- Punch two holes in the back, approximately 70 mm from the left edge of the cardboard sheet (this is where the longest fold-in flap is).
- Insert the elastic band from the outside inward. Make sure you have about 10 mm of elastic band to spare on the inside.
- Insert an eyelet into the pliers and insert the pin with eyelet from the outside through the hole. Arrange the elastic

band so it aligns with the long side of the portfolio. Press firmly in place with pliers.
- Repeat the procedure for the other hole.

Make the cuts for fixing a postcard (150 × 100 mm)

- Open the front cover and flip the portfolio outside up, the front cover being on the right side.
- Place the template on the lower right edge of the front cover.
- Mark the 4 short diagonal lines. It works best with a sharp awl.
- Remove the template and cut the lines precisely.
- Insert the corners of the card.

6. Consideration

The article provides two drawings which enable you to make such portfolios in other sizes. Consider e.g. making a nice wrapping for a present or an invitation. It is also fine to experiment with closings. E.g. a cord could be fixed to the front cover instead of the elastic band and turned around the portfolio. The end is just tucked under the turns of the cord …

The workshop with the participants was a lot of fun, and very recreational as intended by the theme of this year's EuroTEX.

Simple Spreadsheets

Hans Hagen

1. Introduction

Occasionally a question pops up on the ConTEXt mailing list where answering it becomes a nice distraction from a boring task at hand. The spreadsheet module is the result of such a diversion. As with more support code in ConTEXt, this is not a replacement for 'the real thing' but just a nice feature for simple cases. Of course some useful extensions might appear in the future.

2. Spreadsheet tables

We can use Lua in each cell, because under the hood it is all Lua. There is some basic parsing applied so that we can use the usual `A..Z` variables to access cells.

```
\startspreadsheettable[test]
   \startrow
     \startcell 1.1         \stopcell
     \startcell 2.1         \stopcell
     \startcell A[1] + B[1] \stopcell
   \stoprow
   \startrow
     \startcell 2.1         \stopcell
     \startcell 2.2         \stopcell
     \startcell A[2] + B[2] \stopcell
   \stoprow
   \startrow
     \startcell A[1] + B[1] \stopcell
     \startcell A[2] + B[2] \stopcell
     \startcell A[3] + B[3] \stopcell
   \stoprow
\stopspreadsheettable
```

1	3.2
2	4.3
3	7.5

preadsheet.

The rendering is shown in figure 1. Keep in mind that in Lua all calculations are done using floats.
The last cell can also look like this:

```
\startcell
function()
    local s = 0
    for i=1,2 do
        for j=1,2 do
            s = s + dat[i][j]
        end
    end
    return s
end
\stopcell
```

The content of a cell is either a number or a function. In this example we just loop over the (already set) cells and calculate their sum. The `dat` variable accesses the grid of cells.

```
\startcell
function()
    local s = 0
    for i=1,2 do
        for j=1,2 do
            s = s + dat[i][j]
        end
    end
    tmp.total = s
end
\stopcell
```

In this variant we store the sum in the table `tmp` which is local to the current sheet. Another table is `fnc` where we can store functions. This table is shared between all sheets. There are two predefined functions:

```
sum(sometable,firstindex,lastindex)
fmt(specification,n)
```

Let’s see this in action:

```
\startspreadsheettable[test]
   \startrow
     \startcell 1.1 \stopcell
     \startcell 2.1 \stopcell
   \stoprow
   \startrow
     \startcell 2.1 \stopcell
     \startcell 2.2 \stopcell
   \stoprow
   \startrow
     \startcell
        function()
            local s = 0
            for i=1,2 do
                for j=1,2 do
                    s = s + dat[i][j]
                end
            end
            context.bold(s)
        end
     \stopcell
     \startcell
        function()
            local s = 1
            for i=1,2 do
                for j=1,2 do
                    s = s * dat[i][j]
                end
            end
            context.bold(fmt("@.1f",s))
        end
     \stopcell
   \stoprow
\stopspreadsheettable
```

The result is shown in figure 2. Watch the `fmt` call: we use an at sign instead of a percent to please TEX.
Keep in mind that we're typesetting and that doing complex calculations is not our main objective. A typical application of this module is in making bills, for which you can combine it with the correspondence modules. We leave that as an exercise for the reader and stick to a simple example.

2.1
2.2
0.7

e (complex)

```
\startspreadsheettable[test]
  \startrow
    \startcell[align=flushleft,width=8cm] "item one" \stopcell
    \startcell[align=flushright,width=3cm] @ "0.2f EUR" 3.50 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] ”item two” \stopcell
    \startcell[align=flushright] @ ”0.2f EUR” 8.45 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] ”tax 19\percent” \stopcell
    \startcell[align=flushright] @ ”0.2f EUR” 0.19 * (B[1]+B[2])
    \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] ”total 1” \stopcell
    \startcell[align=flushright] @ ”0.2f EUR” sum(B,1,3) \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] ”total 2” \stopcell
    \startcell[align=flushright] @ ”0.2f EUR” B[1] + B[2] + B[3]
    \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] ”total 3” \stopcell
    \startcell[align=flushright] @ ”0.2f EUR” sum(B) \stopcell
  \stoprow
\stopspreadsheettable
```

Here (and in figure 3) you see a quick and more readable way to format cell content. The @ in the template is optional, but needed in cases like this:

```
@ ”(@0.2f) EUR” 8.45
```

A @ is only prepended when no @ is given in the template.

item one	3.50 EUR
item two	8.45 EUR
tax 19%	2.27 EUR
total 1	14.22 EUR
total 2	14.22 EUR
total 3	42.66 EUR

Figure 3: Cells can be formatted by using Q directives.

item one	3.50 EUR
item two	8.45 EUR
tax 19%	2.27 EUR
total 1	14.22 EUR
total 2	14.22 EUR
total 3	42.66 EUR

Figure 4: The `sum` function accumulated stepwise.

In practice this table can be simplified (see figure 4) and made a bit nicer looking.

```
\startspreadsheettable[test][frame=off]
  \startrow
    \startcell[align=flushleft,width=8cm] "The first item" \stopcell
    \startcell[align=flushright,width=3cm] @ "0.2f EUR" 3.50 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "The second item" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" 8.45 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "The third item" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" 5.90 \stopcell
  \stoprow
  \startrow[topframe=on]
    \startcell[align=flushleft] "VAT 19\percent" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" 0.19 * sum(B) \stopcell
  \stoprow
  \startrow[topframe=on]
    \startcell[align=flushleft] "\bf Grand total" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" sum(B) \stopcell
  \stoprow
\stopspreadsheettable
```

There are a few more special start characters. This is demonstrated in figure 5. An `=` character is equivalent to no character and for those who are using regular spreadsheets.[1] When we start with a `!`, the content is not typeset. Strings can be surrounded by single or double quotes and are not really processed.

[1] I must admit that I never used spreadsheets myself, and Taco suggested to support this. However, in the time that we didn't use TEX but used simple ascii based editing we did have summation features built in and they even were part of the early day ConTEXt formats.

The first item	3.50 EUR
The second item	8.45 EUR
The third item	5.90 EUR
VAT 19%	3.39 EUR
Grand total	21.24 EUR

Figure 5: Cells can be hidden by ! and can contain strings only.

```
\startspreadsheettable[test][offset=1ex]
  \startrow
    \startcell[align=flushleft] "first" \stopcell
    \startcell[align=flushleft,width=3cm] '\type{@ "[@i]" 1}'
    \stopcell
    \startcell[align=flushright,width=3cm] @ "[@i]" 1 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "second" \stopcell
    \startcell[align=flushleft] '\type{= 2}' \stopcell
    \startcell[align=flushright] = 2 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "third" \stopcell
    \startcell[align=flushleft] '\type{! 3}' \stopcell
    \startcell[align=flushright] ! 3 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "fourth" \stopcell
    \startcell[align=flushleft] '\type{4}' \stopcell
    \startcell[align=flushright] 4 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "\bf total one" \stopcell
    \startcell[align=flushleft] '\type{sum(C)}' \stopcell
    \startcell[align=flushright] sum(C) \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "\bf total two" \stopcell
    \startcell[align=flushleft] '\type{= sum(C)}' \stopcell
    \startcell[align=flushright] = sum(C) \stopcell
  \stoprow
\stopspreadsheettable
```

The sum function is clever enough not to include itself in the summation. Only preceding cells are taken into account, given that they represent a number.

3. Normal tables

In the previous examples we used TEX commands for structuring the sheet but the content of cells is Lua code. It is also possible to stick to a regular table and use specific commands to set and get cell data.

first	`@ "[@i]" 1`	[1]
second	`= 2`	2
third	`! 3`	
fourth	`4`	4
total one	`sum(C)`	10
total two	`= sum(C)`	20

Figure 6: A sheet can be filled and accessed from regular tables.

```
\bTABLE[align=middle]
   \bTR
     \bTD \getspr{100} \eTD \bTD test \setspr{30} \eTD
   \eTR
   \bTR
     \bTD \getspr{20} \eTD \bTD \getspr{4+3} \eTD
   \eTR
   \bTR
     \bTD \getspr{A[1] + A[2]} \eTD
     \bTD \getspr{B1 + B2} \eTD
   \eTR
   \bTR
     \bTD[nx=2] \bf \getspr{(A[3] + B[3]) /100} \eTD
   \eTR
   \bTR
     \bTD[nx=2] \bf \getspr{fmt("@0.3f",(A[3] + B[3]) /100)} \eTD
   \eTR
   \bTR
     \bTD[nx=2] \bf \getspr{fmt("@0.3f",(sum(A,1,2)) / 10)} \eTD
   \eTR
\eTABLE
```

What method you use depends on the complexity of the table. Of there is more text than data then this method is probably more comfortable.

4. A few settings

It’s possible to influence the rendering. The following example demonstrates this:

```
\bTABLE[align=middle]
   \bTR
     \bTD \getspr{100} \eTD \bTD test \setspr{30} \eTD
   \eTR
   \bTR
     \bTD \getspr{20} \eTD \bTD \getspr{4+3} \eTD
   \eTR
   \bTR
     \bTD \getspr{A[1] + A[2]} \eTD
     \bTD \getspr{B1 + B2} \eTD
   \eTR
   \bTR
     \bTD[nx=2] \bf \getspr{(A[3] + B[3]) /100} \eTD
   \eTR
   \bTR
     \bTD[nx=2] \bf \getspr{fmt("@0.3f",(A[3] + B[3]) /100)} \eTD
   \eTR
   \bTR
     \bTD[nx=2] \bf \getspr{fmt("@0.3f",(sum(A,1,2)) / 10)} \eTD
   \eTR
\eTABLE
```

Figure figure 7 demonstrates how this gets rendered by default. However, often you want numbers to be split in parts separated by periods and commas. This can be done as follows:

123456.78
1234567.89
1358024.67

(large) numbers.

```
\definehighlight[BoldAndRed]  [style=bold,color=darkred]
\definehighlight[BoldAndGreen][style=bold,color=darkgreen]

\setupspreadsheet
  [test]
  [period={\BoldAndRed{.}},
   comma={\BoldAndGreen{,}},
   split=yes]
```

5. The Lua end

You can also use spreadsheets from within Lua. The following example is rather straightforward:

56.78
67.89
24.67

g (large) numbers

```
\startluacode
context.startspreadsheettable { "test" }
    context.startrow()
        context.startcell() context("123456.78")   context.stopcell()
    context.stoprow()
    context.startrow()
        context.startcell() context("1234567.89")  context.stopcell()
    context.stoprow()
    context.startrow()
        context.startcell() context("A[1] + A[2]") context.stopcell()
    context.stoprow()
context.stopspreadsheettable()
\stopluacode
```

However, even more Lua-ish is the next variant:

```
\startluacode
    local set = moduledata.spreadsheets.set
    local get = moduledata.spreadsheets.get

    moduledata.spreadsheets.start("test")
        set("test",1,1,"123456.78")
        set("test",2,1,"1234567.89")
        set("test",3,1,"A[1] + A[2]")
    moduledata.spreadsheets.stop()

    context.bTABLE()
        context.bTR()
            context.bTD() context(get("test",1,1)) context.eTD()
        context.eTR()
        context.bTR()
            context.bTD() context(get("test",2,1)) context.eTD()
        context.eTR()
        context.bTR()
            context.bTD() context(get("test",3,1)) context.eTD()
        context.eTR()
    context.eTABLE()
\stopluacode
```

Of course the second variant does not make much sense as we can do this way more efficient by not using a spreadsheet at all:

```
\startluacode
    local A1, A2 = 123456.78, 1234567.89
    context.bTABLE()
        context.bTR()
            context.bTD() context(A1)    context.eTD()
        context.eTR()
        context.bTR()
            context.bTD() context(A2)    context.eTD()
        context.eTR()
        context.bTR()
            context.bTD() context(A1+A2) context.eTD()
        context.eTR()
    context.eTABLE()
\stopluacode
```

As expected and shown in figure 9, only the first variant gets the numbers typeset nicely.

123,456.78
1,234,567.89
1,358,024.67

123456.78
1234567.89
1358024.67

123456.78
1234567.89
1358024.67

Figure 9: Spreadsheets purely done as ConTEXt Lua Document.

6. Helper macros

There are two helper macros that you can use to see what is stored in a spreadsheet:

```
\inspectspreadsheet[test]
\showspreadsheet   [test]
```

The first command reports the content of `test` to the console, and the second one typesets it in the running text:

```
t={
 { 123456.78, 1234567.89, 1358024.67 },
}
```

Another helper function is `\doifelsespreadsheetcell`, You can use this one to check if a cell is set.

```
(1,1): \doifelsespreadsheetcell[test]{1}{1}{set}{unset}
(2,2): \doifelsespreadsheetcell[test]{2}{2}{set}{unset}
(9,9): \doifelsespreadsheetcell[test]{9}{9}{set}{unset}
```

This gives:

(1,1): set
(2,2): unset
(9,9): unset

There is not much more to say about this module, apart from that it is a nice example of a TEX and Lua mix. Maybe some more (basic) functionality will be added in the future but it all depends on usage.

Oriental TEX: optimizing paragraphs

Hans Hagen & Idris Samawi Hamid

Introduction

One of the objectives of the Oriental TEX project has always been to play with paragraph optimization. The original assumption was that we needed an advanced non-standard paragraph builder to Arabic done right but in the end we found out that a more straightforward approach is to use a sophisticated OpenType font in combination with a paragraph postprocessor that uses the advanced font capabilities. This solution is somewhat easier to imagine than a complex paragraph builder but still involves quite some juggling.

At the June 2012 meeting of the ntg there was a talk about typesetting Devanagari and as fonts are always a nice topic (if only because there is something to show) it made sense to tell a bit more about optimizing Arabic at the same time. In fact, that presentation was already a few years too late because a couple of years back, when the oriental TEX project was presented at tug and Dante meetings, the optimizer was already part of the ConTEXt core code. The main reason for not advocating is was the simple fact that no font other than the (not yet finished) Husayni font provided the relevant feature set.

The lack of advanced fonts does not prevent us from showing what we're dealing with. This is because the ConTEXt mechanisms are generic in the sense that they can also be used with regular Latin fonts, although it does not make that much sense. Anyhow, in the next section we wrap up the current state of typesetting Arabic in ConTEXt. We focus on the rendering, and leave general aspects of bidirectional typesetting and layouts for another time.

This article is written by Idris Samawi Hamid and Hans Hagen and is typeset by ConTEXt MkIV which uses LuaTEX. This program is an extension of TEX that uses Lua to open op the core machinery. The LuaTEX core team consists of Taco Hoekwater, Hartmut Henkel and Hans Hagen.

Manipulating glyphs

When discussing optical optimization of a paragraph, a few alternatives come to mind:

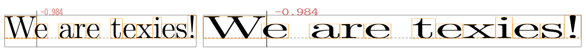
- One can get rid of extensive spaces by adding additional kerns between glyphs. This is often used by poor man's typesetting programs (or routines) and can be applied to non-connecting scripts. It just looks bad. Of course, for connected scripts like Arabic, inter-glyph kerning is not an option, not even in principle.
- Glyphs can be widened a few percent and this is an option that LuaTEX inherits from its predecessor pdfTEX. Normally this goes unnoticed although excessive scaling makes things worse, and yes, one can run into such examples. This

strategy goes under the name hz-optimization (the hz refers to Hermann Zapf, who first came up with this solution).[1]

- A real nice solution is to replace glyphs by narrower or wider variants. This is in fact the ideal hz solution —including for Arabic-script as well— but for it to happen one not only needs needs fonts with alternative shapes, but also a machinery that can deal with them.
- An already old variant is the one first used by Gutenberg, who used alternative cuts for certain combinations of characters. This is comparable with ligatures. However, to make the look and feel optimal, one needs to analyze the text and make decisions on what to replace without loosing consistency.

The solution described here does a bit of everything. As it is mostly meant for a connective script, the starting point is how a scribe works when filling up a line nicely. Depending on how well one can see it coming, the writing can be adapted to widen or narrow following words. And it happens that in Arabic-script there are quite some ways to squeeze more characters in a small area and/or expand some to the extreme to fill up the available space. Shapes can be wider or narrower, they can be stacked and they can get replaced by ligatures. Of course there is some interference with the optional marks on top and below but even there we have some freedom. The only condition is that the characters in a word stay connected.[2]
So, given enough alternative glyphs, one can imagine that excessive interword spacing can be avoided. However, it is non-trivial to check all possible combinations. Actually, it is not needed either, as carefully chosen aesthetic rules put some bounds on what can be done. One should more think in terms of alternative strategies or solutions and this is the terminology that we will therefore use.
Scaling glyphs horizontally is no problem if we keep the scale factor very small, say percentages. This also means that we should not overestimate the impact. For the Arabic script we can stretch more —using non-scaling methods— but again there are some constraints, that we will discuss later on.
In the next example, we demonstrate some excessive stretching:

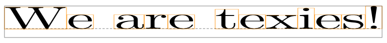
In practice, fonts can provide intercharacter kerning, which is demonstrated next:

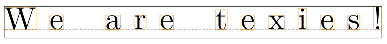


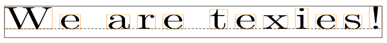
Some poor man's justification routines mess with additional inter-character kerning. Although this is, within reasonable bounds, ok for special purposed like titles, it looks bad in text. The first line expands glyphs and spaces, the second line expands spaces and add additional kerns between characters and the third line expands and add extra kerns.

[1] Sometimes hz-optimization also goes under the rubric of 'Semitic justification'. See, e.g., Bringhurst in pre-3rd editions of his *Elements of Typographic Style*. This technique does not work well for Arabic script in general because glyphs are connected in two dimensions. On the other hand, a certain basic yet ubiquitous Semitic justification *can* be achieved by using the *tawīl* character, commonly called the *kashīdah* (`U+0640`). We will discuss this later in this article.

[2] Much of this is handled within the GPOS features of the OpenType font itself (e.g., `mark` and `mkmk`)







Unfortunately we see quite often examples of the last method in novels and even scientific texts. There is definitely a down side to advanced manipulation.

Applying features to Latin-script

It is easiest is to start out with Latin, if only because it's more intuitive for most of us to see what happens. This is not the place to discuss all the gory details so you have to take some of the configuration options on face value. Once this mechanism is stable and used, the options can be described. For now we stick to presenting the idea.

Let's assume that you know what font features are. The idea is to work with combinations of such features and figure out what combination suits best. In order not to clutter a document style, these sets are defined in so called goodie files. Here is an excerpt of `demo.lfg`:

```
return {
  name = "demo",
  version = "1.01",
  comment = "An example of goodies.",
  author = "Hans Hagen",
  featuresets = {
    simple = {
      mode   = "node",
      script = "latn"
    },
    default = {
      mode   = "node",
      script = "latn",
      kern   = "yes",
    },
    ligatures = {
      mode   = "node",
      script = "latn",
      kern   = "yes",
      liga   = "yes",
    },
    smallcaps = {
      mode   = "node",
      script = "latn",
      kern   = "yes",
      smcp   = "yes",
    },
  },
  solutions = {
    experimental = {
      less = {
        "ligatures", "simple",
      },
      more = {
        "smallcaps",
      },
    },
  },
}
```

We see four sets of features here. You can use these sets in a ConTEXt feature definition, like:

```
\definefontfeature
  [solution-demo]
  [goodies=demo,
   featureset=default]
```

You can use a set as follows:

```
\definefont
  [SomeTestFont]
  [texgyrepagellaregular*solution-demo at 10pt]
```

So far, there is nothing special or new, but we can go a step further.

```
\definefontsolution
  [solution-a]
  [goodies=demo,
   solution=experimental,
   method={normal,preroll},
   criterium=1]

\definefontsolution
  [solution-b]
  [goodies=demo,
   solution=experimental,
   method={normal,preroll,split},
   criterium=1]
```

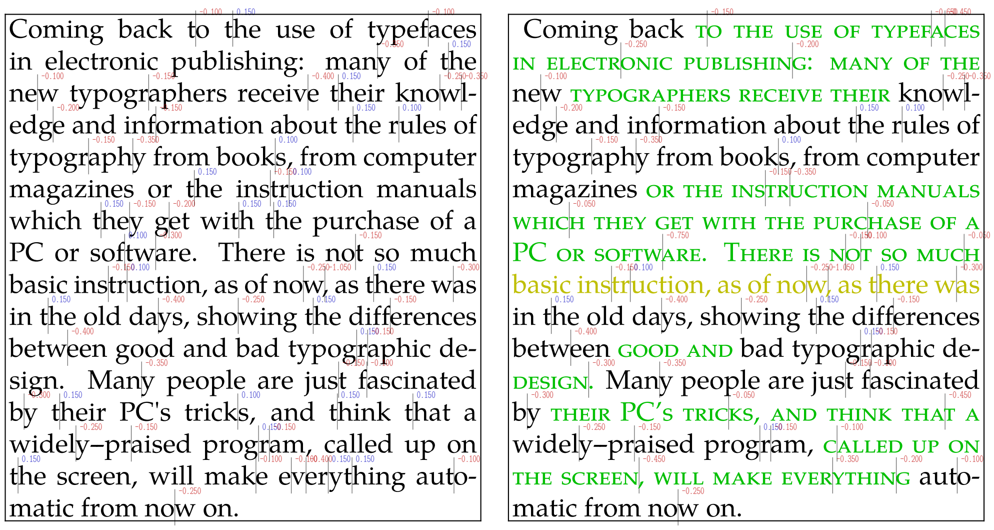
Here we have defined two solutions. They refer to the `experimental` solution in the goodie file `demo.lfg`. A solution has a `less` and a `more` entry. The featuresets mentioned there reflect ways to make a word narrower or wider. There can be more than one way to do that, although it comes at a performance price. Before we see how this works out we turn on a tracing option:

```
\enabletrackers
  [builders.paragraphs.solutions.splitters.colors]
```

This will color the words in the result according to what has happened. When a featureset out of the `more` category has been applied, the words turn green, when `less` is applied, the word becomes yellow. The `preroll` option in the `method` list makes sure that we do a more extensive test beforehand.

```
\SomeTestFont \startfontsolution[solution-a]
\input zapf \par
\stopfontsolution
```

In Figure 1 we see what happens. In each already split line words get wider or narrower until we're satisfied. A criterium of 1 is pretty strict[3]. Keep in mind that we use some arbitrary features here. We try removing kerns to get narrower although there is nothing that guarantees that kerns are positive. On the other hand, using ligatures might help. In order to get wider we use smallcaps. Okay, the result will look somewhat strange but so does much typesetting nowadays.



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

normal

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

solution

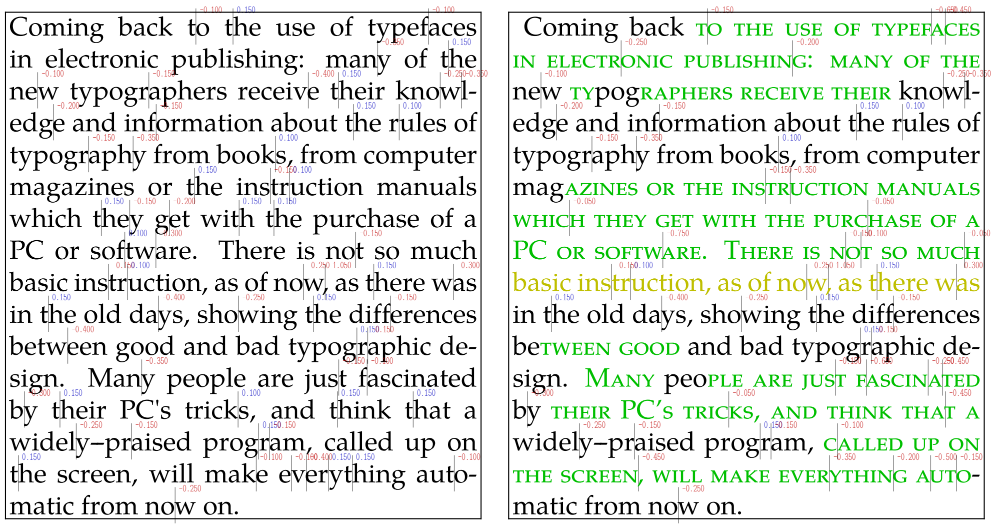
Figure 1: Solution a.

[3] This number reflects the maximum badness and future versions might have a different measure with more granularity.

There is one pitfall here. This mechanism is made for a connective script where hyphenation is not used. As a result a word here is actually split up when it has discretionaries and of course this text fragment has. It goes unnoticed in the rendering but is of course far from optimal.

```
\SomeTestFont \startfontsolution[solution-b]
\input zapf \par
\stopfontsolution
```

In this example (Figure 2) we keep words as a whole but as a side effect we skip words that are broken across a line. This is mostly because it makes not much sense to implement it as Latin is not our target. Future versions of ConTEXt might get more sophisticated font machinery so then things might look better.



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

normal

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

solution

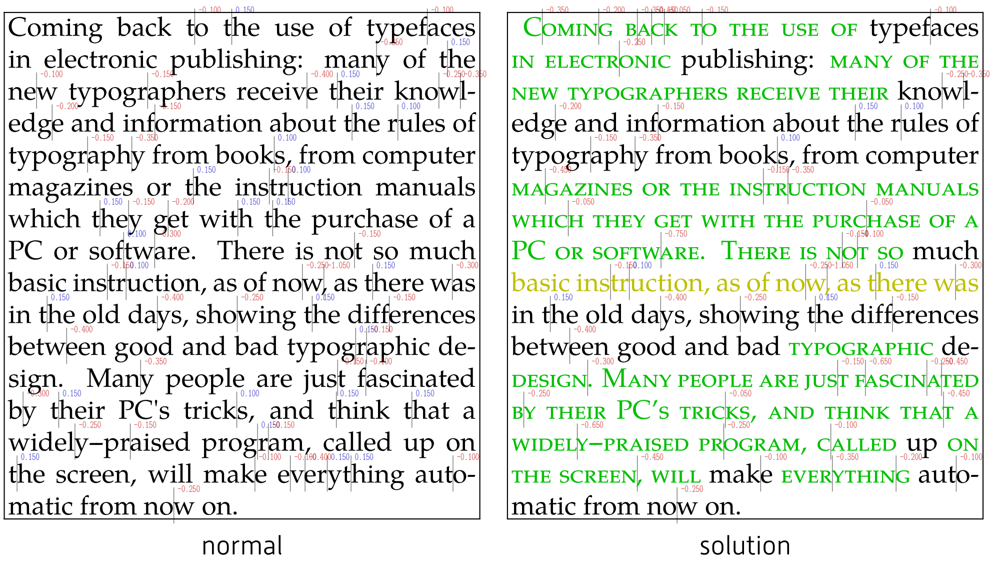
Figure 2: Solution b.

We show two more methods:

```
\definefontsolution
  [solution-c]
  [goodies=demo,
   solution=experimental,
   method={reverse,preroll},
   criterium=1]
```

```
\definefontsolution
  [solution-d]
  [goodies=demo,
   solution=experimental,
   method={random,preroll,split},
   criterium=1]
```

In Figure 3 we start at the other end of a line. As we sort of mimick a scribe, we can be one who plays safe at the start of corrects at the end.


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

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

Figure 3: Solution c.

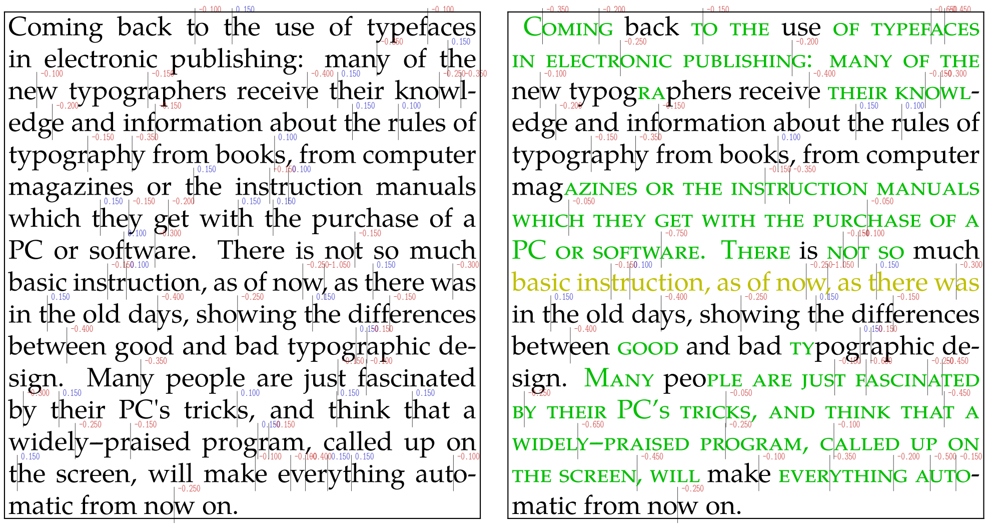
In Figure 4 we add some randomness but to what extent this works well depends on how many words we need to retypeset before we get the badness of the line within the constraints.

Salient features of Arabic-script

Before applying the above to Arabic-script, let’s discuss some salient aspects of the problem. As a cursive script, Arabic is extremely versatile and the scribal calligraphy tradition reflects that. Digital Arabic typography is only beginning to catch up with the possibilities afforded by the scribal tradition. Indeed, early lead-punch typography and typesetting of Arabic-script was more advanced than most digital typography even up to this day. In any case, let us begin to organize some of that versatility into a taxonomy for typography purposes.

What's available?

We have to work within the following parameters:



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

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

Figure 4: Solution d.

- No hyphenation ever (well, almost never)

  It is commonly pointed out that there is no hyphenation is Arabic. This is something of a half-truth. In the manuscript tradition one actually does find something akin to hyphenation. In the ancient Kufic script, breaking a word across lines is actually quite common. But even in the more modern Naskh script, the one most normal Arabic text fonts are based on, it does occur, albeit rarely and presumably when the scribe is out of options for the line he is working on. Indeed, one could regard it as a failure on the part of the scribe once he reaches the end of the line.[4]
  But there is still an important rule, regardless of whether we use Naskh, Kufic, or any other Arabic script. Consider the word below:

  الفاعل

  It is a single word composed of two cursive strings. One could actually hyphenate it, with our rule being to break it at the end of the first cursive string and before the beginning of the second cursive string:

  هذا اسم الفا
  عل

[4] Indeed, even Latin hyphenation, when it occurs, can be considered a ‘failure’ of sorts.

Again, it's a rare phenomenon and hardly ever occurs in modern typesetting, lead-punch or digital, if at all. On the other hand, it could have some creative uses in future Arabic-script typography.

- Macrotypography (aesthetic features)

  In Arabic there are often numerous aesthetic ways of writing out the exact same semantic string:[5]

  الحمد الحمد الحمد

  Normally we put combine OpenType features into feature sets that are each internally and aesthetically coherent. So in the above example we have used three different sets, reading from right to left. We'll call them simple, default, and dipped.
  Just as Latin typography uses separate fonts to mark off different uses of text (bold, italic, etc.), an advanced Arabic font can use aesthetic feature sets to similar effect. This works best on distinguishing long streams of text from one another, since the differences between feature sets are not always noticeable on short strings. That is, two different aesthetic sets may type a given short string, such as a single word, exactly the same way. Consider the above three sets (simple, default, and dipped) once more:

  علي علي علي

  For the above string the default and dipped aesthetic sets (middle and left) give the exact same result, while the basic one (right) remains, well, quite basic.
  Let's go back to our earlier example:

  الحمد الحمد الحمد

  Note that the simple version is wider than the default, and the dipped version is (slightly) thinner than the default. This relates to another point: An aesthetic feature set can serve two functions:

  1. It can serve as the base aesthetic style.
  2. It can serve as a resource for glyph substitution for a given string in another base aesthetic style.

  This brings us back to our main topic.

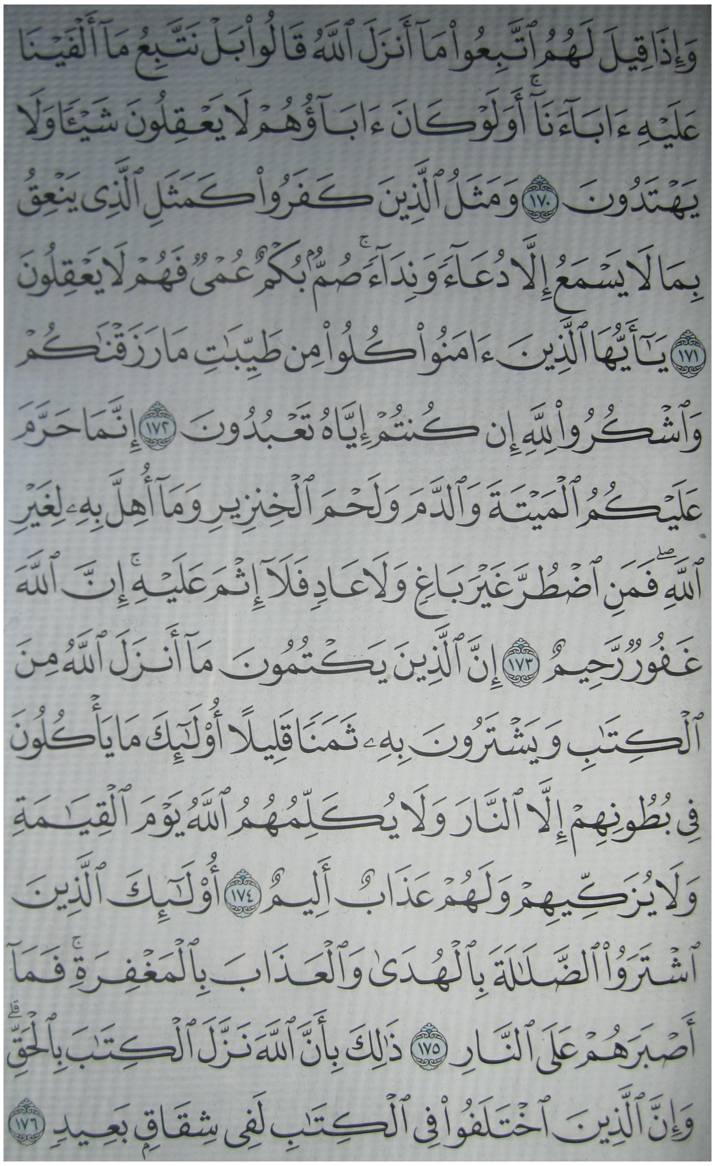
[5] This five character string can be represented in Latin by the five character string '*al-md*' (not including the '-'). It is pronounced '*al-amdu*'. Note that Arabic script is mainly consonantal: pure vowels are not part of the alphabet and are, instead, represented by diacritics.

- Microtypography (paragraph optimization features)

  Here our job is to optimize the paragraph for even spacing and aesthetic viewing. It turns out that there are a number of ways to look at this issue, and we will begin exploring these in the next subsection.

Two approaches

Let us start off with a couple of samples. Qurānic transcription has always been the gold standard of Arabic-script. In Figure 5 we see a nice example of scribal optimization. The scribe here is operating under the constraint that each page ends with the end



وَإِذَا قِيلَ لَهُمُ ٱتَّبِعُوا۟ مَآ أَنزَلَ ٱللَّهُ قَالُوا۟ بَلْ نَتَّبِعُ مَآ أَلْفَيْنَا
عَلَيْهِ ءَابَآءَنَآ ۗ أَوَلَوْ كَانَ ءَابَآؤُهُمْ لَا يَعْقِلُونَ شَيْـًٔا وَلَا
يَهْتَدُونَ ﴿١٧٠﴾ وَمَثَلُ ٱلَّذِينَ كَفَرُوا۟ كَمَثَلِ ٱلَّذِى يَنْعِقُ
بِمَا لَا يَسْمَعُ إِلَّا دُعَآءً وَنِدَآءً ۚ صُمٌّۢ بُكْمٌ عُمْىٌ فَهُمْ لَا يَعْقِلُونَ
﴿١٧١﴾ يَـٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ كُلُوا۟ مِن طَيِّبَـٰتِ مَا رَزَقْنَـٰكُمْ
وَٱشْكُرُوا۟ لِلَّهِ إِن كُنتُمْ إِيَّاهُ تَعْبُدُونَ ﴿١٧٢﴾ إِنَّمَا حَرَّمَ
عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ ٱلْخِنزِيرِ وَمَآ أُهِلَّ بِهِۦ لِغَيْرِ
ٱللَّهِ ۖ فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ فَلَآ إِثْمَ عَلَيْهِ ۚ إِنَّ ٱللَّهَ
غَفُورٌ رَّحِيمٌ ﴿١٧٣﴾ إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ أَنزَلَ ٱللَّهُ مِنَ
ٱلْكِتَـٰبِ وَيَشْتَرُونَ بِهِۦ ثَمَنًا قَلِيلًا ۙ أُو۟لَـٰٓئِكَ مَا يَأْكُلُونَ
فِى بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ ٱلْقِيَـٰمَةِ
وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ أَلِيمٌ ﴿١٧٤﴾ أُو۟لَـٰٓئِكَ ٱلَّذِينَ
ٱشْتَرَوُا۟ ٱلضَّلَـٰلَةَ بِٱلْهُدَىٰ وَٱلْعَذَابَ بِٱلْمَغْفِرَةِ ۚ فَمَآ
أَصْبَرَهُمْ عَلَى ٱلنَّارِ ﴿١٧٥﴾ ذَٰلِكَ بِأَنَّ ٱللَّهَ نَزَّلَ ٱلْكِتَـٰبَ بِٱلْحَقِّ ۗ
وَإِنَّ ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى ٱلْكِتَـٰبِ لَفِى شِقَاقٍۭ بَعِيدٍ ﴿١٧٦﴾

Figure 5: Scribal Optimization. Scribe: *Uthmān āhā*. Qurān, circa 1997.

of a Qurānic verse (designated by the symbol U+06DD ۝). That is, no verse is broken across pages. That constraint, which is by no means mandatory or universal, gives the scribe lots of space for optimization, even more than normal.
In Figure 6 we have a page of the famous *al-Husayni Muaf* of 1919–1923, which remains up to this day the only typeset copy of the Qurān to attain general acceptance in the Muslim world. Indeed, it remains the standard 'edition' of the Qurān and even later scribal copies, such as the one featured in Figure 5 are based on its orthography. Unlike the scribal version, the typesetters of the *al-Husayni Muaf* did not try to constrain each page to end with the end of a Qurānic verse. Again, that is a nice feature to have as it makes recitation somewhat easier but it is by no means a mandatory one.
In any case, both samples share verses 172–176 in commmon, so there is lots to compare and contrast. We will also use these verses as our main textual sample for paragraph optimization.
Using Figure 5 and Figure 6 as benchmarks, we can begin by analyzing the approaches to paragraph optimization in Arabic-script typography into two kinds:

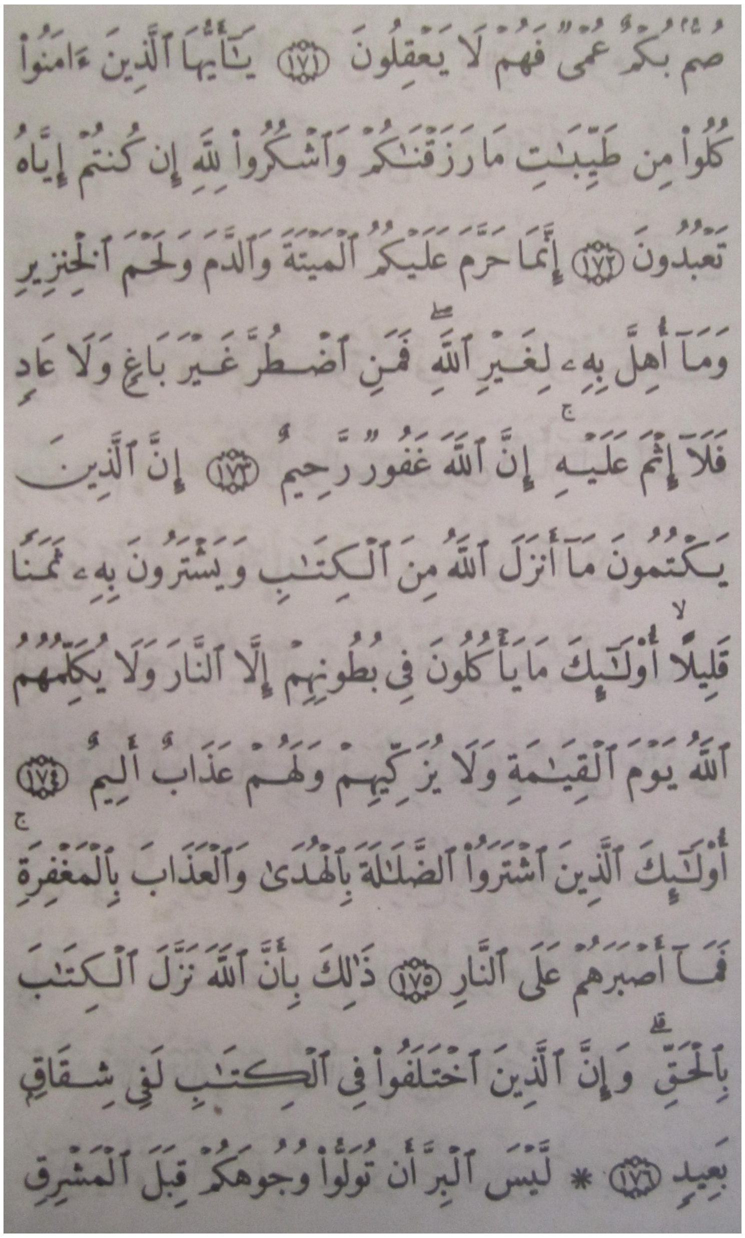
- Alternate glyphs

  Much of pre-digital Arabic typography uses this method. Generally, a wide variant of a letter is used to take up the space which would normally get absorbed by hyphenation in Latin. Here are examples of three of the most common substitutions, again, reading from right to left:

  ى ← الّذين   الكتاب ← الكتاب   ما ← ما

  Each of the six strings above occurs in Figure 6. Identifying them is an exercise left to the reader. We call these kinds of alternate glyphs *alternate-shaped* glyphs. The three substitutions above are the most common alternate-glyph substitutions found in pre-digital Arabic-script typography, including some contextual variants (initial, medial, final, and isolated) where appropriate. (The scribal tradition contains a lot more alternate-shaped glyphs. A few lead-punch fonts implement some of them, and we have implemented many of these in our Husayni font.) The results generally look quite nice and much more professional than most digital Arabic typography, which generally dispenses with these alternates.
  But one also finds attempts at *extending* individual characters without changing the shape very much. One finds this already in Figure 6. We call these kinds of alternate glyphs *naturally curved widened* glyphs, or just *naturally widened* glyphs for short. Sometimes this is done for the purpose of making enough space for the vowels (which in Arabic take the form of diacritic characters). For example:

  يأيها ← يَٰٓأَيُّهَا



صُمٌّۢ بُكْمٌ عُمْىٌ فَهُمْ لَا يَعْقِلُونَ ﴿١٧١﴾ يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟
كُلُوا۟ مِن طَيِّبَٰتِ مَا رَزَقْنَٰكُمْ وَٱشْكُرُوا۟ لِلَّهِ إِن كُنتُمْ إِيَّاهُ
تَعْبُدُونَ ﴿١٧٢﴾ إِنَّمَا حَرَّمَ عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ ٱلْخِنزِيرِ
وَمَآ أُهِلَّ بِهِۦ لِغَيْرِ ٱللَّهِ ۖ فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ
فَلَآ إِثْمَ عَلَيْهِ ۚ إِنَّ ٱللَّهَ غَفُورٌ رَّحِيمٌ ﴿١٧٣﴾ إِنَّ ٱلَّذِينَ
يَكْتُمُونَ مَآ أَنزَلَ ٱللَّهُ مِنَ ٱلْكِتَٰبِ وَيَشْتَرُونَ بِهِۦ ثَمَنًا
قَلِيلًا ۙ أُو۟لَٰٓئِكَ مَا يَأْكُلُونَ فِى بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ
ٱللَّهُ يَوْمَ ٱلْقِيَٰمَةِ وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ أَلِيمٌ ﴿١٧٤﴾
أُو۟لَٰٓئِكَ ٱلَّذِينَ ٱشْتَرَوُا۟ ٱلضَّلَٰلَةَ بِٱلْهُدَىٰ وَٱلْعَذَابَ بِٱلْمَغْفِرَةِ ۚ
فَمَآ أَصْبَرَهُمْ عَلَى ٱلنَّارِ ﴿١٧٥﴾ ذَٰلِكَ بِأَنَّ ٱللَّهَ نَزَّلَ ٱلْكِتَٰبَ
بِٱلْحَقِّ ۗ وَإِنَّ ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى ٱلْكِتَٰبِ لَفِى شِقَاقٍۭ
بَعِيدٍ ﴿١٧٦﴾ ۞ لَّيْسَ ٱلْبِرَّ أَن تُوَلُّوا۟ وُجُوهَكُمْ قِبَلَ ٱلْمَشْرِقِ

Figure 6: Alternate Fixed Glyphs. From the *al-Husayni Muaf* of the Qurān, 1923.

As you can see, there are two letters that have been *widened* for vowel accommodation. In Figure 6 there are some good but near-clumsy attempts at this. We say, 'near-clumsy' because the typographers and typesetters mix natural, curved, *widened* variants of letters with flat, horizontal, *extended* versions. One reason for this is that a full repertoire of naturally curved glyph alternates would be much too unwieldy for even the best lead-punch typesetting machines and their operators. Even with these limitations one can find brave examples of lead-based typesetting that do a good job of sophisticated paragraph optimization via glyph

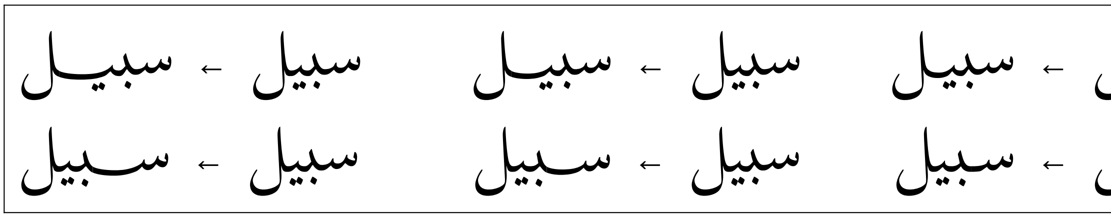
alternates, both widened and alternate-shaped. Figure 7 is a representative example (in the context of columns).
Careful examination of this two-column sample will reveal the tension between naturally *widened* and horizontally *extended* glyphs in the execution of paragraph optimization. On the other hand, there is one apparent 'rule' that one finds in this and other examples of lead-punch Arabic-script typesetting:

> *Generally, there is only one naturally widened character per word or one alternate-shaped character per word.*

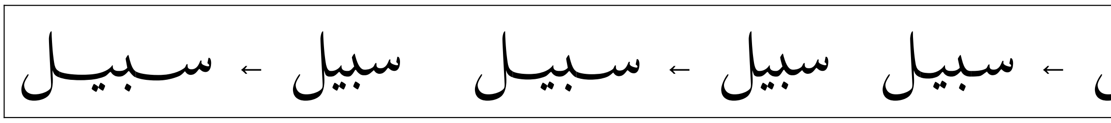
In Figure 5 one can see that this 'rule' is not always observed by scribes, see, e.g., the middle word in line 9 from the top, which uses two of the alternate-shaped characters we encountered above (can you identify that word?). But we still need some constraints for decent-looking typesetting, and the above tentative rule is a good place to start the analysis. For widened characters in particular we see that even the scribe (Figure 5) closely approximates this rule. So let's begin improving on our tentative rule somewhat, and expand it into a number of possibilities. Let's look at the naturally-widened-glyph case first:

> *Generally, there is only one naturally widened character allowed per word. However, two extended non-consecutive characters may be allowed.* (The logic of the experimental font Husayni already has contraints that prevent *consecutive* curved widened characters).

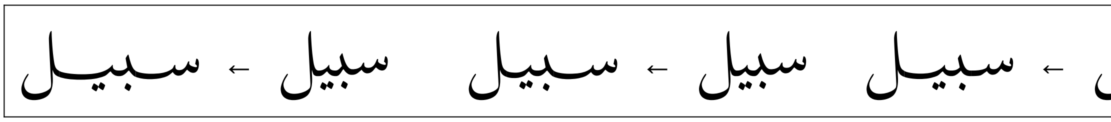
For example, we prefer to get widening like the following:



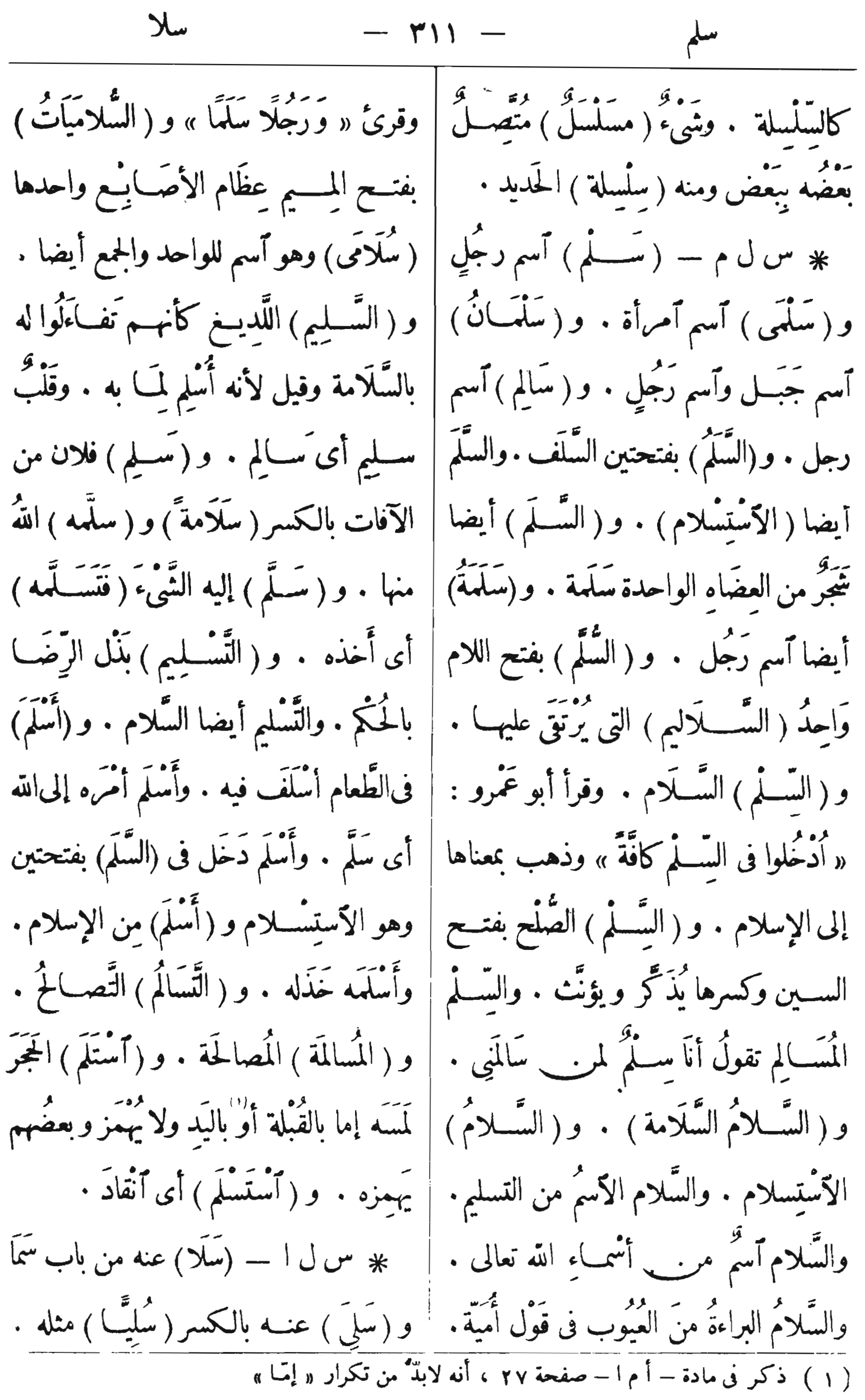
But as, e.g., a last resort or for stylistic purposes we can also do



Or even better, we mix it up a bit. That is, if there is more than one widened character, one should be longer than the other, e.g.:



One will notice that the middle substitution (where the first widened character is longer than the second) does not look as good as the two outer ones (where the second is longer than the first). These kinds of aesthetic issues can be formalized for future work. In the meantime, here is a working modified version of the rule for naturally-widened-glyphs:



سلم — ٣١١ — سلا

كالسِّلْسِلة . وشَىْءٌ (مسَلْسَلٌ) مُتَّصِلٌ
بَعْضُه بِبَعْض ومنه (سِلْسِلة) الحَديد .
* س ل م — (سَلْم) آسم رجُلٍ
و (سَلْمَى) آسم آمرأة . و (سَلْمَانُ)
آسم جَبَل وآسم رَجُلٍ . و (سَالِم) آسم
رجل . و (السَّلَمُ) بفتحتين السَّلَف . والسَّلَم
أيضا (الآسْتِسْلام) . و (السَّلَم) أيضا
شَجَرٌ من العِضَاهِ الواحدة سَلَمة . و (سَلَمَةُ)
أيضا آسم رَجُل . و (السُّلَّم) بفتح اللام
وَاحِدُ (السَّلَالِيم) التى يُرْتَقَى عليها .
و (السِّلْم) السَّلَام . وقرأ أبو عَمْرو :
« أُدْخُلوا فى السِّلْمِ كافَّةً » وذهب بمعناها
إلى الإسلام . و (السِّلْم) الصُّلْح بفتح
السين وكسرها يُذَكَّر ويؤنَّث . والسِّلْم
المُسَالِم تقولُ أنَا سِلْمٌ لمن سَالَمَنِي .
و (السَّلامُ السَّلَامة) . و (السَّلامُ)
الآسْتِسلام . والسَّلام الآسمُ من التسليم .
والسَّلام آسمٌ من أسْماءِ الله تعالى .
والسَّلامُ البراءةُ منَ العُيُوب فى قَوْل أُمَيّة .

وقرِئ « وَرَجُلًا سَلَمًا » و (السُّلامَيَاتُ)
بفتح المِيم عِظَام الأصَابِع واحدها
(سُلَامَى) وهو آسم للواحد والجمع أيضا .
و (السَّلِيم) اللَّدِيغ كأنهم تَفاءَلُوا له
بالسَّلَامة وقيل لأنه أُسْلِم لِمَا به . وقَلْبٌ
سلِيم أى سَالِم . و (سَلِم) فلان من
الآفات بالكسر (سَلَامةً) و (سلَّمه) اللهُ
منها . و (سَلَّم) إليه الشَّىْءَ (فَتَسَلَّمه)
أى أَخذه . و (التَّسْلِيم) بَذْل الرِّضَا
بالحُكْم . والتَّسْليم أيضا السَّلام . و (أَسْلَمَ)
فى الطَّعام أسْلَفَ فيه . وأَسْلَمَ أمْرَه إلى الله
أى سَلَّم . وأَسْلَمَ دَخَل فى (السَّلَم) بفتحتين
وهو الآستِسْلام و (أَسْلَمَ) مِن الإسلام .
وأَسْلَمَه خَذَله . و (التَّسَالُم) التَّصالُحُ .
و (المُسالَمَة) المُصالحَة . و (آسْتَلَمَ) الحَجَر
لَمَسَه إما بالقُبْلة أو[1] باليَدِ ولا يُهْمَز وبعضهم
يَهمِزه . و (آسْتَسْلَمَ) أى آنْقادَ .
* س ل ا — (سَلَا) عنه من باب سَمَا
و (سَلِيَ) عنه بالكسر (سُلِيًّا) مثله .

(١) ذكر فى مادة — أ م ا — صفحة ٢٧ ، أنه لابدّ من تكرار « إمّا »

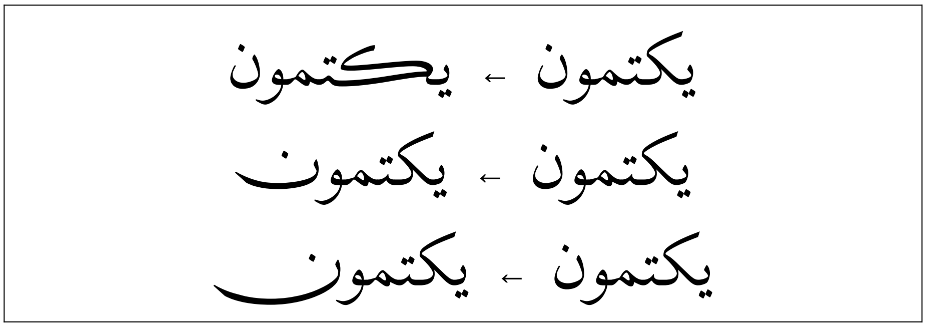
Figure 7: Mixed Alternate Glyphs in Two Columns. From the classical dictionary *Mukhtār al-iā*.

> *Generally, there is only one naturally widened character allowed per word. However, two non-consecutive widened characters may be allowed. In that case, the second widened character should be longer than the first.*

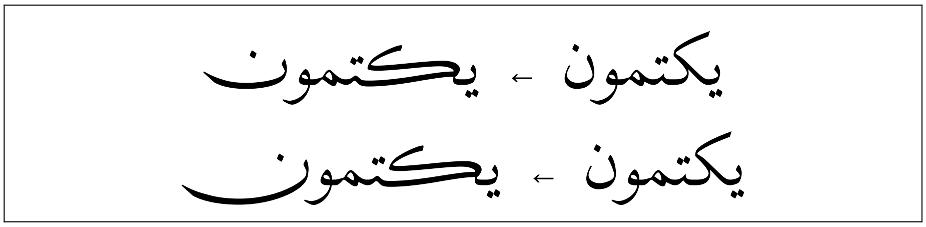
One case where cases of two naturally widened character will be common is in poetry, which involves wide lines. We'll say more about this in the section on flat extending.
Now let's look at the alternate-shaped case:

> *Generally, there is only one alternate-shaped character allowed per word. However, two non-consecutive alternate-shaped characters may be allowed.*

So we prefer, e.g.,



but we could have, e.g, as a last resort or as a stylistic option,

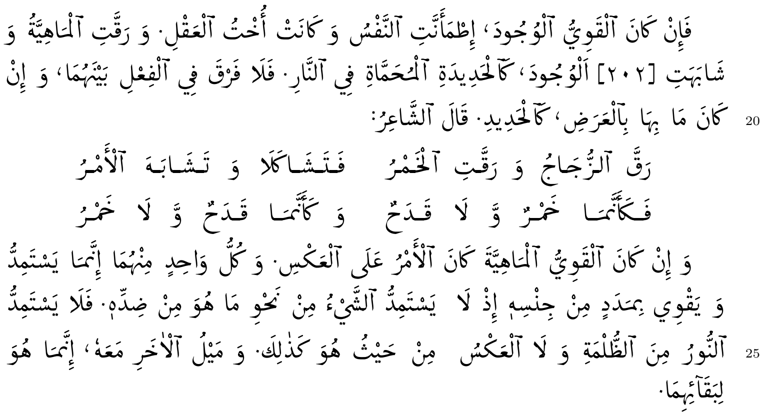


Again, in poetry this kind of multiple substitution within a single word could occur frequently. A challenge will be to develop a system of parameters where we can almost predict which kinds of substitution will happen under a given set of values of those parameters.

- Flat extending

In the transition from lead-punch to digital typography, alternate-glyph substitution largely vanished.[6] The problem of spacing remained, and a simple yet inelegant solution was adopted: flat, horizontal extending of characters. Now this solution did have *some* precedent in pre-digital Arabic typography, as you can see in Figure 6 and Figure 7. This solution had the advantage that it required only a single character: a simple horizontal bar called a *tawīl* or more commonly

[6] Indeed, as was the case with Latin typography, Arabic-script typography took a sharp turn for the worse with the advent of digital typography. On the other hand, Latin typography recovered much more quickly, in large part thanks to Knuth's development of TEX.


فَإِنْ كَانَ ٱلْقَوِيُّ ٱلْوُجُودَ، إِطْمَأَنَّتِ ٱلنَّفْسُ وَ كَانَتْ أُخْتُ ٱلْعَقْلِ. وَ رَقَّتِ ٱلْمَاهِيَّةُ وَ
شَابَهَتِ [٢٠٢] ٱلْوُجُودَ، كَٱلْحَدِيدَةِ ٱلْمُحَمَّاةِ فِي ٱلنَّارِ. فَلَا فَرْقَ فِي ٱلْفِعْلِ بَيْنَهُمَا، وَ إِنْ
كَانَ مَا بِهَا بِٱلْعَرَضِ، كَٱلْحَدِيدِ. قَالَ ٱلشَّاعِرُ:

رَقَّ ٱلـزُّجَاجُ وَ رَقَّـتِ ٱلْخَـمْـرُ فَـتَـشَـاكَـلَا وَ تَـشَـابَـهَ ٱلْأَمْـرُ
فَـكَأَنَّمَـا خَـمْـرٌ وَّ لَا قَـدَحٌ وَ كَأَنَّمَـا قَـدَحٌ وَّ لَا خَـمْـرُ

وَ إِنْ كَانَ ٱلْقَوِيُّ ٱلْمَاهِيَّةَ كَانَ ٱلْأَمْرُ عَلَى ٱلْعَكْسِ. وَ كُلٌّ وَاحِدٍ مِنْهُمَا إِنَّمَا يَسْتَمِدُّ
وَ يَقْوِي بِمَدَدٍ مِنْ جِنْسِهِ إِذْ لَا يَسْتَمِدُّ ٱلشَّيْءُ مِنْ نَحْوِ مَا هُوَ مِنْ ضِدِّهِ. فَلَا يَسْتَمِدُّ
ٱلنُّورُ مِنَ ٱلظُّلْمَةِ وَ لَا ٱلْعَكْسُ مِنْ حَيْثُ هُوَ كَذٰلِكَ. وَ مَيْلُ ٱلْآخَرِ مَعَهُ، إِنَّمَا هُوَ
لِبَقَآئِهِمَا.

Figure 8: Poetry Justification in ArabTEX.

a *kashidah* (U+0640). This character could then be repeated as often as necessary to fill any extra space.

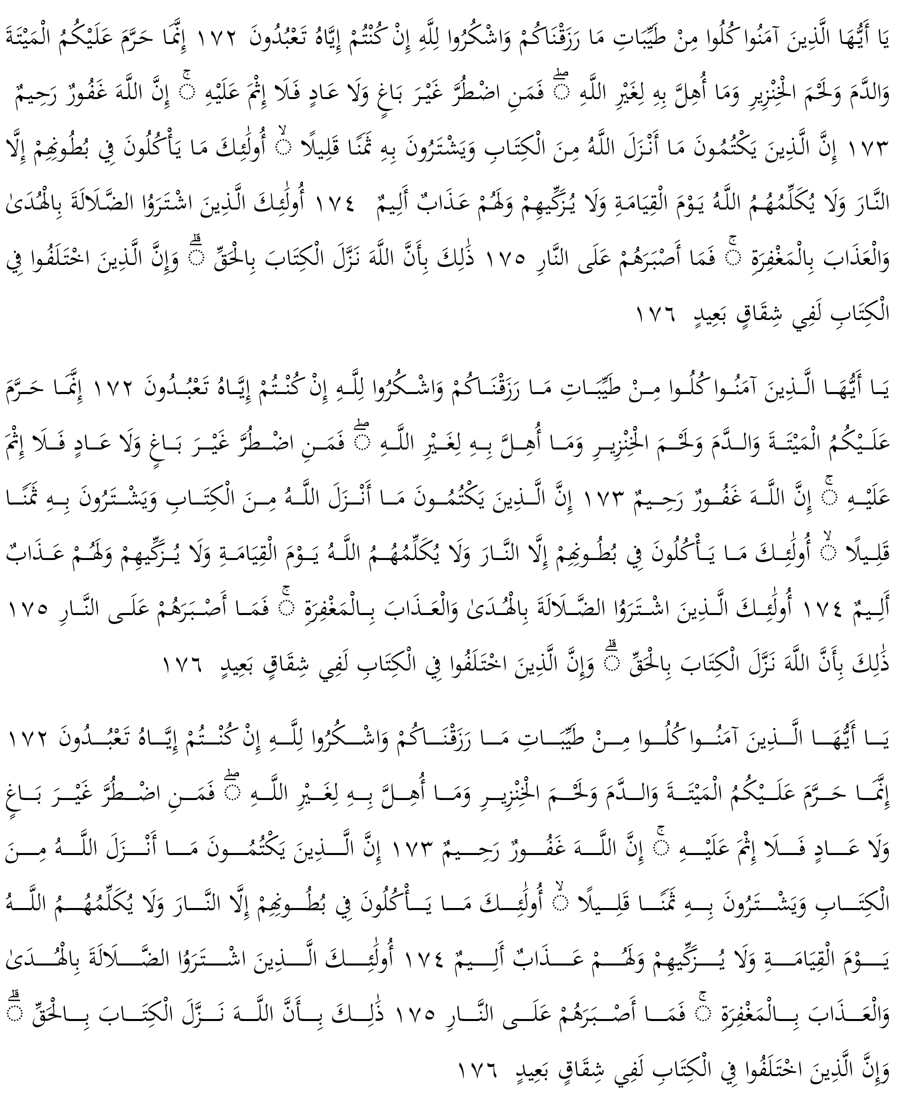
Now an examination of pre-digital books shows a (rather wise) reticence to using this method too slavishly. That reticence has now been thrown to the winds. This can be seen by looking at the standard implementation of flat extending as provided by Microsoft Word. This program provides three levels of extending that it calls 'justification'. See Figure 9 for examples of all three. The minimum level is actually very close to the default (i.e., no-justification) level. Note that the sample text used in Figure 9 is the same as that used in the earlier samples from the Qurān.

Older implementations of Arabic-script within TEX, such as ArabTEX and Omega/Aleph, also provided facilities for flat extending. The most common use was in poetry, which requires a fixed width for each stanza.

In Omega/Aleph, a method based on `\xleaders` was used, based on a very thin *tawīl* glyph (much thinner than U+0640) that could be used for very fine extending optimization based on TEX's badness parameter. One nice application is in marginal notes: See Figure 10, where the marginal note on the right is zoomed in. On the other hand, we see that the leaders method creates extending that may be considered too perfectly even: Do we want to impose the rule that only one character should be extended per word (or at most two non-consecutive characters)? I have seen a lot of older digital Arabic typography that does even extending, including the poetry in the ArabTEX sample in figure 8. Compare this with the Microsoft Word method (Figure 9). The method used in Microsoft Word, with only one extension per word, seems to be the current standard for flat-extending justification.

On the other hand, the justification used in Microsoft Word is not particularly aesthetically pleasing. The answer will lie, again, in parameterization of some sort to be determined. As TEXies, we want to be able to have fine control over this kind of behavior in any case. In the meantime, we mirror the same rule we arrived at for naturally-widened-glyphs:

Generally, there is only one flat extended character allowed per word. However, two non-consecutive extended characters may be allowed. In that case, the second extended character should be longer than the first.


يَا أَيُّهَا الَّذِينَ آمَنُوا كُلُوا مِنْ طَيِّبَاتِ مَا رَزَقْنَاكُمْ وَاشْكُرُوا لِلَّهِ إِنْ كُنْتُمْ إِيَّاهُ تَعْبُدُونَ ١٧٢ إِنَّمَا حَرَّمَ عَلَيْكُمُ الْمَيْتَةَ
وَالدَّمَ وَلَحْمَ الْخِنْزِيرِ وَمَا أُهِلَّ بِهِ لِغَيْرِ اللَّهِ ۖ فَمَنِ اضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ فَلَا إِثْمَ عَلَيْهِ ۚ إِنَّ اللَّهَ غَفُورٌ رَحِيمٌ
١٧٣ إِنَّ الَّذِينَ يَكْتُمُونَ مَا أَنْزَلَ اللَّهُ مِنَ الْكِتَابِ وَيَشْتَرُونَ بِهِ ثَمَنًا قَلِيلًا ۙ أُولَٰئِكَ مَا يَأْكُلُونَ فِي بُطُونِهِمْ إِلَّا
النَّارَ وَلَا يُكَلِّمُهُمُ اللَّهُ يَوْمَ الْقِيَامَةِ وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ أَلِيمٌ ١٧٤ أُولَٰئِكَ الَّذِينَ اشْتَرَوُا الضَّلَالَةَ بِالْهُدَىٰ
وَالْعَذَابَ بِالْمَغْفِرَةِ ۚ فَمَا أَصْبَرَهُمْ عَلَى النَّارِ ١٧٥ ذَٰلِكَ بِأَنَّ اللَّهَ نَزَّلَ الْكِتَابَ بِالْحَقِّ ۗ وَإِنَّ الَّذِينَ اخْتَلَفُوا فِي
الْكِتَابِ لَفِي شِقَاقٍ بَعِيدٍ ١٧٦

يَا أَيُّهَا الَّـذِينَ آمَنُوا كُلُـوا مِـنْ طَيِّبَـاتِ مَـا رَزَقْنَـاكُمْ وَاشْكُرُوا لِلَّـهِ إِنْ كُنْـتُمْ إِيَّـاهُ تَعْبُـدُونَ ١٧٢ إِنَّمَـا حَـرَّمَ
عَلَـيْكُمُ الْمَيْتَـةَ وَالـدَّمَ وَلَحْـمَ الْخِنْزِيـرِ وَمَـا أُهِـلَّ بِـهِ لِغَـيْرِ اللَّـهِ ۖ فَمَـنِ اضْـطُرَّ غَيْـرَ بَـاغٍ وَلَا عَـادٍ فَـلَا إِثْمَ
عَلَيْـهِ ۚ إِنَّ اللَّـهَ غَفُـورٌ رَحِـيمٌ ١٧٣ إِنَّ الَّـذِينَ يَكْتُمُـونَ مَـا أَنْـزَلَ اللَّـهُ مِـنَ الْكِتَـابِ وَيَشْـتَرُونَ بِـهِ ثَمَنًـا
قَلِـيلًا ۙ أُولَٰئِـكَ مَـا يَـأْكُلُونَ فِي بُطُـونِهِمْ إِلَّا النَّـارَ وَلَا يُكَلِّمُهُـمُ اللَّـهُ يَـوْمَ الْقِيَامَـةِ وَلَا يُـزَكِّيهِمْ وَلَهُـمْ عَـذَابٌ
أَلِـيمٌ ١٧٤ أُولَٰئِـكَ الَّـذِينَ اشْـتَرَوُا الضَّـلَالَةَ بِالْهُـدَىٰ وَالْعَـذَابَ بِـالْمَغْفِرَةِ ۚ فَمَـا أَصْـبَرَهُمْ عَلَـى النَّـارِ ١٧٥
ذَٰلِكَ بِأَنَّ اللَّهَ نَزَّلَ الْكِتَابَ بِالْحَقِّ ۗ وَإِنَّ الَّذِينَ اخْتَلَفُوا فِي الْكِتَابِ لَفِي شِقَاقٍ بَعِيدٍ ١٧٦

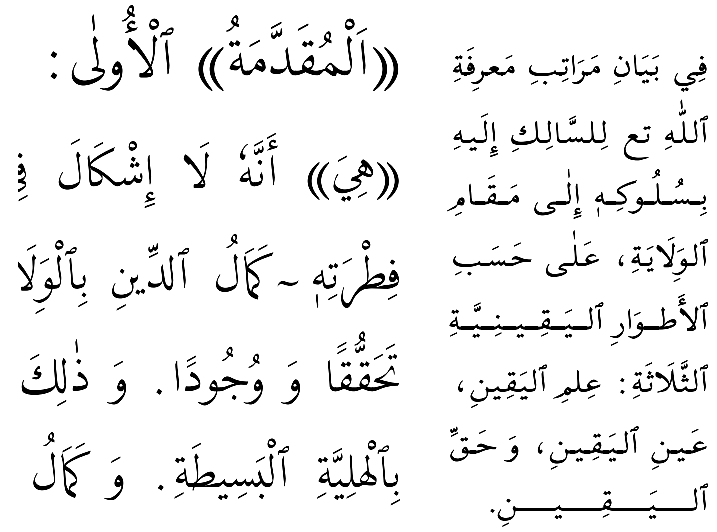
يَـا أَيُّهَـا الَّـذِينَ آمَنُـوا كُلُـوا مِـنْ طَيِّبَـاتِ مَـا رَزَقْنَـاكُمْ وَاشْـكُرُوا لِلَّـهِ إِنْ كُنْـتُمْ إِيَّـاهُ تَعْبُـدُونَ ١٧٢
إِنَّمَـا حَـرَّمَ عَلَـيْكُمُ الْمَيْتَـةَ وَالـدَّمَ وَلَحْـمَ الْخِنْزِيـرِ وَمَـا أُهِـلَّ بِـهِ لِغَـيْرِ اللَّـهِ ۖ فَمَـنِ اضْـطُرَّ غَيْـرَ بَـاغٍ
وَلَا عَـادٍ فَـلَا إِثْمَ عَلَيْـهِ ۚ إِنَّ اللَّـهَ غَفُـورٌ رَحِـيمٌ ١٧٣ إِنَّ الَّـذِينَ يَكْتُمُـونَ مَـا أَنْـزَلَ اللَّـهُ مِـنَ
الْكِتَـابِ وَيَشْـتَرُونَ بِـهِ ثَمَنًـا قَلِـيلًا ۙ أُولَٰئِـكَ مَـا يَـأْكُلُونَ فِي بُطُـونِهِمْ إِلَّا النَّـارَ وَلَا يُكَلِّمُهُـمُ اللَّـهُ
يَـوْمَ الْقِيَامَـةِ وَلَا يُـزَكِّيهِمْ وَلَهُـمْ عَـذَابٌ أَلِـيمٌ ١٧٤ أُولَٰئِـكَ الَّـذِينَ اشْـتَرَوُا الضَّـلَالَةَ بِالْهُـدَىٰ
وَالْعَـذَابَ بِـالْمَغْفِرَةِ ۚ فَمَـا أَصْـبَرَهُمْ عَلَـى النَّـارِ ١٧٥ ذَٰلِـكَ بِـأَنَّ اللَّـهَ نَـزَّلَ الْكِتَـابَ بِـالْحَقِّ ۗ
وَإِنَّ الَّذِينَ اخْتَلَفُوا فِي الْكِتَابِ لَفِي شِقَاقٍ بَعِيدٍ ١٧٦

Figure 9: Flat justification from Microsoft Word 2010.

For example:

سبيل ← سبيـل
سبيل ← سـبيـــل

In accordance with our working rule, the top substitution uses only one flat extended character. The bottom uses two, but the second is longer than the first.

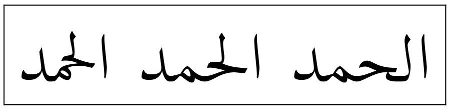


((اَلْمُقَدَّمَةُ)) ٱلْأُولَىٰ :
((هِيَ)) أَنَّهُ لَا إِشْكَالَ فِي
فِطْرَتِهِ ~ كَمَالُ ٱلدِّينِ بِٱلْوَلَا
تَحَقُّقًا وَ وُجُودًا . وَ ذٰلِكَ
بِٱلْهَلِيَّةِ ٱلْبَسِيطَةِ. وَ كَمَالُ

فِي بَيَانِ مَرَاتِبِ مَعرِفَةِ
ٱللّٰهِ تع لِلسَّالِكِ إِلَيهِ
بِسُلُوكِهِ إِلَى مَقَامِ
ٱلوَلَايَةِ، عَلَىٰ حَسَبِ
ٱلأَطوَارِ ٱليَقِينِيَّةِ
ٱلثَّلَاثَةِ: عِلمِ ٱليَقِينِ،
عَينِ ٱليَقِينِ، وَ حَقِّ
ٱليَقِينِ.

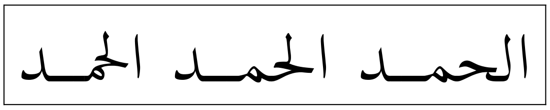
Figure 10: Marginal-note justification in Omega/Aleph.

In our own estimation, the smaller the type, as in, e.g., footnotes and marginal notes, the less aesthetic variants that are needed. And the less aesthetic variants needed, the better that flat extending will work as a solution. Consider another example of the same word processed in three different variants:



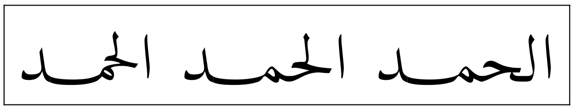
الحمد الحمد الحمد

In this case our default is on the left. The variant on the right is about as basic as one can get; the default on the left is a sophisticated aesthetic variant. The middle one is, well, in between. Let's try them with flat extending, using only one extended character per word:



الحمـــد الحمـــد الحمـــد

On the left, we have an aesthetic combination of letters followed by a flat *tawīl*. This is what Microsoft Word would give us, and the result is aesthetically distasteful. In the word on the right, however, the flat extension fits well with the basic nature of the feature set. As for the middle one, it could go either way and we leave it to the reader to decide what one thinks.
Now let's repeat with more naturally curved widening:



الحمـــد الحمـــد الحمـــد

Here, the variant on the left comes out much nicer. The one on the right looks okay with curved widening, although one could arguably do better with flat extending, at least in some contexts. The middle one, again, could go either way, though we think it does somewhat better with curved widening compared to the one on the right. The variant on the left only works well with curved widening.

Towards a ConTEXt solution

In what follows, we will focus on a solution to the problem of paragraph optimization via alternate glyphs (including alternately-shaped and naturally-widened variants). It turns out that the `\xleaders` method used by Omega/Aleph does not work in LuaTEX, so flat extending could not be naively implemented that way. At the moment flat extending is yet to be implemented in ConTEXt.
Since flat extending is so ubiquitous in current Arabic-script typography, and since it does have important applications (poetry and small font sizes where one prefers simpler aesthetic variants), one could ask why this was not implemented first. In part, this is because the immediate priority of the Oriental TEX project has been top-notch, unparalleled aesthetic sophistication of the script. As we noted above, flat extending does not work so well with sophisticated aesthetic variation. So although the flat-extending problem is apparently simpler, it is understandable that we have focused on the more difficult problem first. A clear understanding of the issues and challenges involved with the more general alternate glyph method will help us implement a solution to the the flat-extended problem as a special case. We will come back to this issue towards the end.
Let us now consider the current experimental ConTEXt setup for paragraph optimization for Arabic-script.

Applying Features to Arabic-script

We're now ready for the real thing: Arabic script. The initial setup is not that different from the Latin-script case.

```
\definefontfeature
  [husayni-whatever]
  [goodies=husayni,
   featureset=default]

\definefontsolution
  [FancyHusayni]
  [goodies=husayni,
   solution=experimental]

\definefont
  [FancyHusayni]
  [file:husayni*husayni-whatever at 24pt]
```

But here the definitions in the goodies file look way more complex. Here we have only one shrink set but multiple expansion sets.

```
local yes = "yes"
local basics = {
    analyze  = yes,
    mode     = "node",
    language = "dflt",
    script   = "arab",
}
local analysis = {
    ccmp = yes,
    init = yes, medi = yes, fina = yes,
}
local regular = {
   rlig = yes, calt = yes, salt = yes, anum = yes,
   ss01 = yes, ss03 = yes, ss07 = yes, ss10 = yes, ss12 = yes,
   ss15 = yes, ss16 = yes, ss19 = yes, ss24 = yes, ss25 = yes,
   ss26 = yes, ss27 = yes, ss31 = yes, ss34 = yes, ss35 = yes,
   ss36 = yes, ss37 = yes, ss38 = yes, ss41 = yes, ss42 = yes,
   ss43 = yes, js16 = yes,
}
local positioning = {
   kern = yes, curs = yes, mark = yes, mkmk = yes,
}
local minimal_stretching = {
    js11 = yes, js03 = yes,
}
local medium_stretching = {
    js12=yes, js05=yes,
}
local maximal_stretching= {
    js13 = yes, js05 = yes, js09 = yes,
}
local wide_all = {
    js11 = yes, js12 = yes, js13 = yes, js05 = yes, js09 = yes,
}
local shrink = {
    flts = yes, js17 = yes, ss05 = yes, ss11 = yes, ss06 = yes,
    ss09 = yes,
}
local default = {
    basics, analysis, regular, positioning,
}
```

```
return {
  name = "husayni",
  version = "1.00",
  comment = "Goodies that complement the Husayni font by prof.Hamid.",
  author = "Idris Samawi Hamid and Hans Hagen",
  featuresets = {
    default = {
      default,
    },
    minimal_stretching = {
      default,
      js11 = yes, js03 = yes,
    },
    medium_stretching = {
      default,
      js12=yes, js05=yes,
    },
    maximal_stretching= {
      default,
      js13 = yes, js05 = yes, js09 = yes,
    },
    wide_all = {
      default,
      js11 = yes, js12 = yes, js13 = yes, js05 = yes, js09 = yes,
    },
    shrink = {
      default, flts = yes,
      js17 = yes,
      ss05 = yes, ss11 = yes, ss06 = yes, ss09 = yes,
    },
   },
  solutions = {
    experimental = {
      less = {
        "shrink",
      },
      more = {
        "minimal_stretching", "medium_stretching", "maximal_stretching",
        "wide_all"
      },
    },
  },
  ...
}
```

There are some 55 stylistic and 21 justification features. Not all make sense when optimizing. We predefine some Lua tables to make the sets and solutions easier to understand. The default rendering looks as follows:

```
\FancyHusayni
\righttoleft
\definefontfeature[rasm][script=arab,ss05=yes,js06=no,ss55=yes]
\addff{rasm}
\getbuffer[sample]
\par
```

يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ كُلُوا۟ مِن طَيِّبَٰتِ مَا رَزَقْنَٰكُمْ وَٱشْكُرُوا۟ لِلَّهِ إِن كُنتُمْ
إِيَّاهُ تَعْبُدُونَ ﴿١٧٢﴾ إِنَّمَا حَرَّمَ عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ ٱلْخِنزِيرِ وَمَآ أُهِلَّ
بِهِۦ لِغَيْرِ ٱللَّهِ ۖ فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ فَلَآ إِثْمَ عَلَيْهِ ۚ إِنَّ ٱللَّهَ غَفُورٌ
رَّحِيمٌ ﴿١٧٣﴾ إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ أَنزَلَ ٱللَّهُ مِنَ ٱلْكِتَٰبِ وَيَشْتَرُونَ بِهِۦ
ثَمَنًا قَلِيلًا ۙ أُو۟لَٰٓئِكَ مَا يَأْكُلُونَ فِى بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ
ٱلْقِيَٰمَةِ وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ أَلِيمٌ ﴿١٧٤﴾ أُو۟لَٰٓئِكَ ٱلَّذِينَ ٱشْتَرَوُا۟ ٱلضَّلَٰلَةَ
بِٱلْهُدَىٰ وَٱلْعَذَابَ بِٱلْمَغْفِرَةِ ۚ فَمَآ أَصْبَرَهُمْ عَلَى ٱلنَّارِ ﴿١٧٥﴾ ذَٰلِكَ بِأَنَّ
ٱللَّهَ نَزَّلَ ٱلْكِتَٰبَ بِٱلْحَقِّ ۗ وَإِنَّ ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى ٱلْكِتَٰبِ لَفِى شِقَاقٍۭ
بَعِيدٍ ﴿١٧٦﴾

Note that we already have a degree of widened substitution in this example. This is all for the accommodation of vowels, and is defined entirely in the OpenType tables of the font. We also added some special orthography (the `rasm` font feature to get the Qurānic features just right). You can also do this by adding the feature to the `lfg` file (`local regular =`). There is no paragraph optimization as yet, although the default LuaTEX engine does a good job to start with.
Next we show a more optimized result:

```
\setupfontsolution
  [FancyHusayni]
  [method={preroll,normal},
   criterium=1]

\startfontsolution[FancyHusayni]
  \FancyHusayni
  \righttoleft
\definefontfeature[rasm][script=arab,ss05=yes,js06=no,ss55=yes]
\addff{rasm}
  \getbuffer[sample]
  \par
\stopfontsolution
```

يَـٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ كُلُوا۟ مِن طَيِّبَـٰتِ مَا رَزَقْنَـٰكُمْ وَٱشْكُرُوا۟ لِلَّهِ إِن كُنتُمْ
إِيَّاهُ تَعْبُدُونَ ﴿١٧٢﴾ إِنَّمَا حَرَّمَ عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ ٱلْخِنزِيرِ وَمَآ أُهِلَّ
بِهِۦ لِغَيْرِ ٱللَّهِ ۖ فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ فَلَآ إِثْمَ عَلَيْهِ ۚ إِنَّ ٱللَّهَ غَفُورٌ
رَّحِيمٌ ﴿١٧٣﴾ إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ أَنزَلَ ٱللَّهُ مِنَ ٱلْكِتَـٰبِ وَيَشْتَرُونَ بِهِۦ
ثَمَنًا قَلِيلًا ۙ أُو۟لَـٰٓئِكَ مَا يَأْكُلُونَ فِى بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ
ٱلْقِيَـٰمَةِ وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ أَلِيمٌ ﴿١٧٤﴾ أُو۟لَـٰٓئِكَ ٱلَّذِينَ ٱشْتَرَوُا۟ ٱلضَّلَـٰلَةَ
بِٱلْهُدَىٰ وَٱلْعَذَابَ بِٱلْمَغْفِرَةِ ۚ فَمَآ أَصْبَرَهُمْ عَلَى ٱلنَّارِ ﴿١٧٥﴾ ذَٰلِكَ بِأَنَّ
ٱللَّهَ نَزَّلَ ٱلْكِتَـٰبَ بِٱلْحَقِّ ۗ وَإِنَّ ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى ٱلْكِتَـٰبِ لَفِى شِقَاقٍۭ
بَعِيدٍ ﴿١٧٦﴾

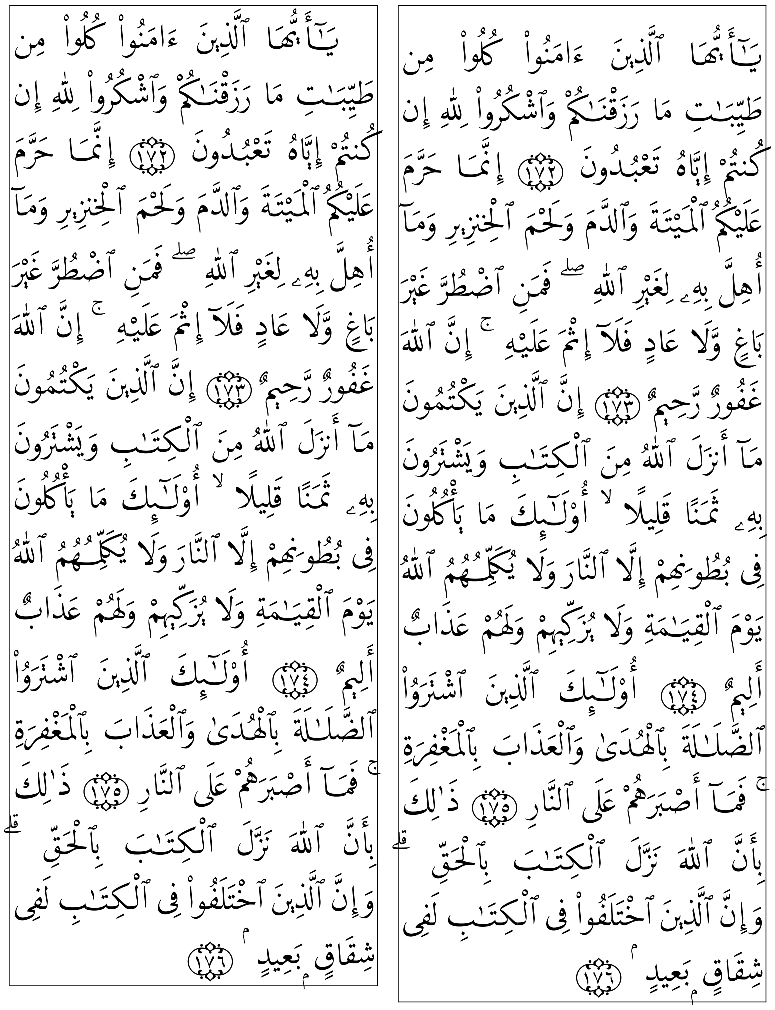
Now let’s see what happens when `\parfillskip = 0pt`, i.e., the last line has no extra space after the end of the paragraph. This is important for getting, e.g., the last line of the page to end with the end of a verse as we discussed earlier:

```
\setupfontsolution
  [FancyHusayni]
  [method={preroll,normal},
   criterium=1]

\startfontsolution[FancyHusayni]
  \FancyHusayni
  \righttoleft
\definefontfeature[rasm][script=arab,ss05=yes,js06=no,ss55=yes]
\addff{rasm}
  \parfillskip=0pt
  \getbuffer[sample]
  \par
\stopfontsolution
```

يَـٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ كُلُوا۟ مِن طَيِّبَـٰتِ مَا رَزَقْنَـٰكُمْ وَٱشْكُرُوا۟ لِلَّهِ إِن
كُنتُمْ إِيَّاهُ تَعْبُدُونَ ١٧٢ إِنَّمَا حَرَّمَ عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ ٱلْخِنزِيرِ
وَمَآ أُهِلَّ بِهِۦ لِغَيْرِ ٱللَّهِ ۖ فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ فَلَآ إِثْمَ
عَلَيْهِ ۚ إِنَّ ٱللَّهَ غَفُورٌ رَّحِيمٌ ١٧٣ إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ أَنزَلَ
ٱللَّهُ مِنَ ٱلْكِتَـٰبِ وَيَشْتَرُونَ بِهِۦ ثَمَنًا قَلِيلًا ۙ أُو۟لَـٰٓئِكَ مَا يَأْكُلُونَ فِى
بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ ٱلْقِيَـٰمَةِ وَلَا يُزَكِّيهِمْ وَلَهُمْ
عَذَابٌ أَلِيمٌ ١٧٤ أُو۟لَـٰٓئِكَ ٱلَّذِينَ ٱشْتَرَوُا۟ ٱلضَّلَـٰلَةَ بِٱلْهُدَىٰ وَٱلْعَذَابَ
بِٱلْمَغْفِرَةِ ۚ فَمَآ أَصْبَرَهُمْ عَلَى ٱلنَّارِ ١٧٥ ذَٰلِكَ بِأَنَّ ٱللَّهَ نَزَّلَ ٱلْكِتَـٰبَ
بِٱلْحَقِّ ۗ وَإِنَّ ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى ٱلْكِتَـٰبِ لَفِى شِقَاقٍۭ بَعِيدٍ ١٧٦

Just as the effects are more visible in the `\parfillskip = 0pt` case, the impact is much larger when the available width is less. In figures 11, 12, 13, 14 and 15 we can see the optimizer in action when that happens.
In our estimation, the current experimental solution works best for alternate-shaped glyphs, although there is some success with naturally widened characters. Clearly, some widened substitutions work better than others. A lot of fine tuning is needed, both within the OpenType features as well as the optimization algorithm.



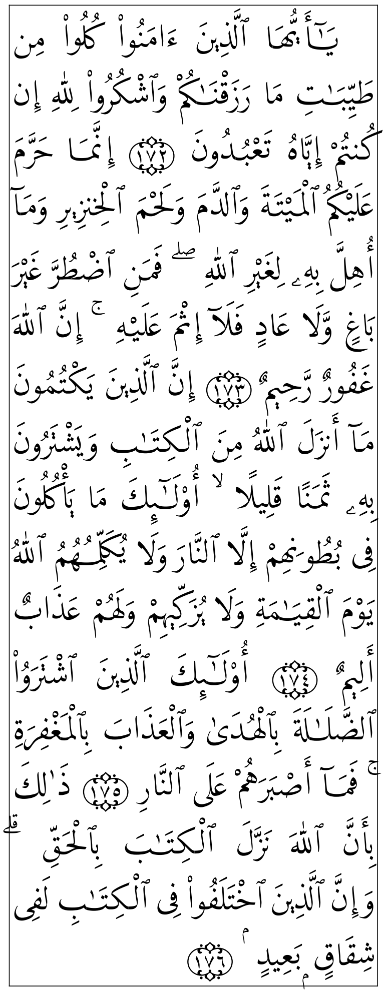
يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُواْ كُلُواْ مِن
طَيِّبَٰتِ مَا رَزَقۡنَٰكُمۡ وَٱشۡكُرُواْ لِلَّهِ إِن
كُنتُمۡ إِيَّاهُ تَعۡبُدُونَ ١٧٢ إِنَّمَا حَرَّمَ
عَلَيۡكُمُ ٱلۡمَيۡتَةَ وَٱلدَّمَ وَلَحۡمَ ٱلۡخِنزِيرِ وَمَآ
أُهِلَّ بِهِۦ لِغَيۡرِ ٱللَّهِۖ فَمَنِ ٱضۡطُرَّ غَيۡرَ
بَاغٖ وَلَا عَادٖ فَلَآ إِثۡمَ عَلَيۡهِۚ إِنَّ ٱللَّهَ
غَفُورٞ رَّحِيمٌ ١٧٣ إِنَّ ٱلَّذِينَ يَكۡتُمُونَ
مَآ أَنزَلَ ٱللَّهُ مِنَ ٱلۡكِتَٰبِ وَيَشۡتَرُونَ
بِهِۦ ثَمَنٗا قَلِيلًاۙ أُوْلَٰٓئِكَ مَا يَأۡكُلُونَ
فِي بُطُونِهِمۡ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ ٱللَّهُ
يَوۡمَ ٱلۡقِيَٰمَةِ وَلَا يُزَكِّيهِمۡ وَلَهُمۡ عَذَابٌ
أَلِيمٌ ١٧٤ أُوْلَٰٓئِكَ ٱلَّذِينَ ٱشۡتَرَوُاْ
ٱلضَّلَٰلَةَ بِٱلۡهُدَىٰ وَٱلۡعَذَابَ بِٱلۡمَغۡفِرَةِۚ
فَمَآ أَصۡبَرَهُمۡ عَلَى ٱلنَّارِ ١٧٥ ذَٰلِكَ
بِأَنَّ ٱللَّهَ نَزَّلَ ٱلۡكِتَٰبَ بِٱلۡحَقِّۗ
وَإِنَّ ٱلَّذِينَ ٱخۡتَلَفُواْ فِي ٱلۡكِتَٰبِ لَفِي
شِقَاقِۭ بَعِيدٖ ١٧٦

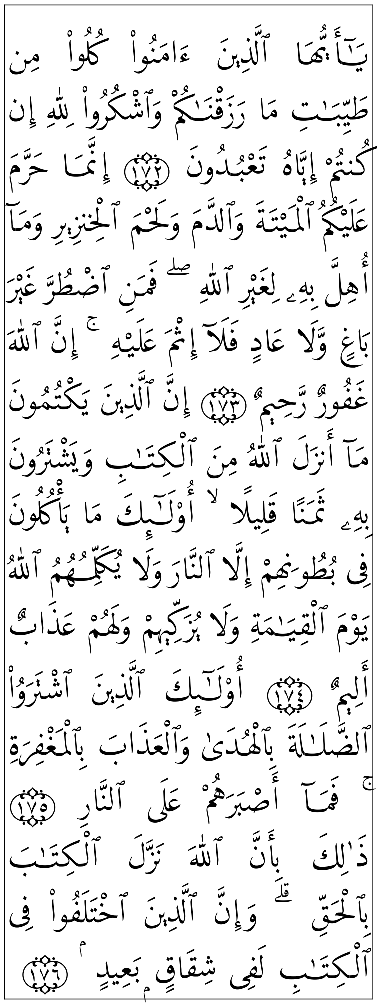
normal

يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُواْ كُلُواْ مِن
طَيِّبَٰتِ مَا رَزَقۡنَٰكُمۡ وَٱشۡكُرُواْ لِلَّهِ إِن
كُنتُمۡ إِيَّاهُ تَعۡبُدُونَ ١٧٢ إِنَّمَا حَرَّمَ
عَلَيۡكُمُ ٱلۡمَيۡتَةَ وَٱلدَّمَ وَلَحۡمَ ٱلۡخِنزِيرِ وَمَآ
أُهِلَّ بِهِۦ لِغَيۡرِ ٱللَّهِۖ فَمَنِ ٱضۡطُرَّ غَيۡرَ
بَاغٖ وَلَا عَادٖ فَلَآ إِثۡمَ عَلَيۡهِۚ إِنَّ ٱللَّهَ
غَفُورٞ رَّحِيمٌ ١٧٣ إِنَّ ٱلَّذِينَ يَكۡتُمُونَ
مَآ أَنزَلَ ٱللَّهُ مِنَ ٱلۡكِتَٰبِ وَيَشۡتَرُونَ
بِهِۦ ثَمَنٗا قَلِيلًاۙ أُوْلَٰٓئِكَ مَا يَأۡكُلُونَ
فِي بُطُونِهِمۡ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ ٱللَّهُ
يَوۡمَ ٱلۡقِيَٰمَةِ وَلَا يُزَكِّيهِمۡ وَلَهُمۡ عَذَابٌ
أَلِيمٌ ١٧٤ أُوْلَٰٓئِكَ ٱلَّذِينَ ٱشۡتَرَوُاْ
ٱلضَّلَٰلَةَ بِٱلۡهُدَىٰ وَٱلۡعَذَابَ بِٱلۡمَغۡفِرَةِۚ
فَمَآ أَصۡبَرَهُمۡ عَلَى ٱلنَّارِ ١٧٥ ذَٰلِكَ
بِأَنَّ ٱللَّهَ نَزَّلَ ٱلۡكِتَٰبَ بِٱلۡحَقِّۗ
وَإِنَّ ٱلَّذِينَ ٱخۡتَلَفُواْ فِي ٱلۡكِتَٰبِ لَفِي
شِقَاقِۭ بَعِيدٖ ١٧٦

narrow

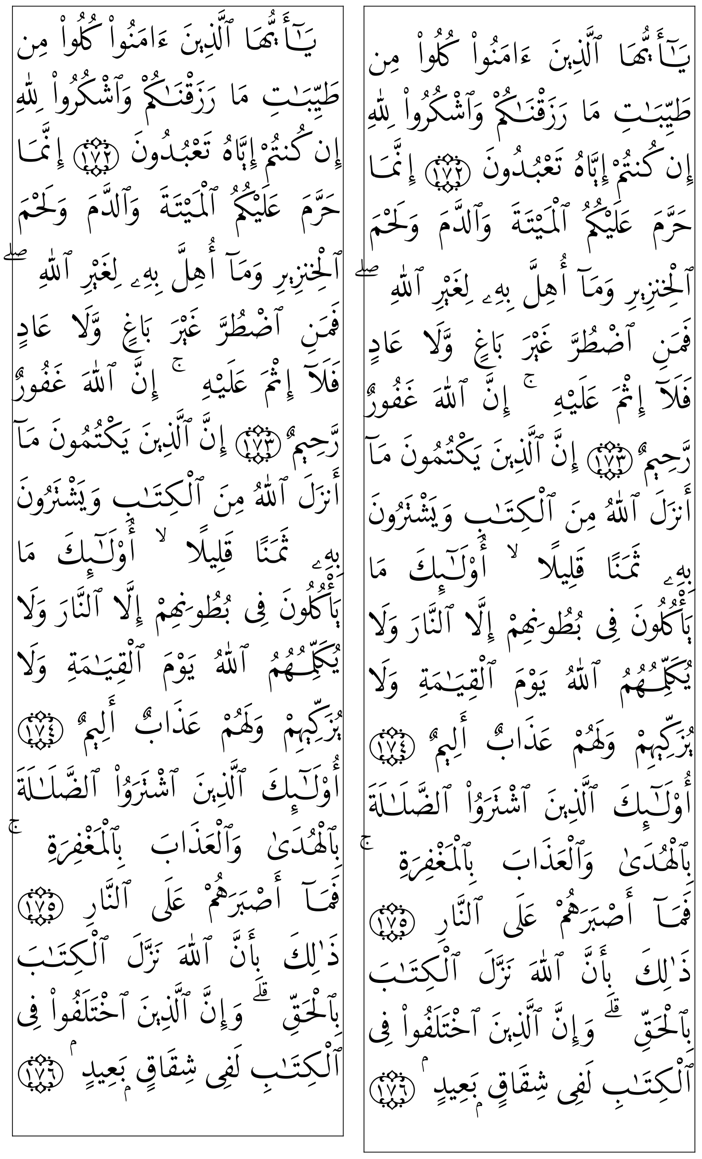
Figure 11: A narrower sample (a).


يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُواْ كُلُواْ مِن
طَيِّبَٰتِ مَا رَزَقۡنَٰكُمۡ وَٱشۡكُرُواْ لِلَّهِ إِن
كُنتُمۡ إِيَّاهُ تَعۡبُدُونَ ١٧٢ إِنَّمَا حَرَّمَ
عَلَيۡكُمُ ٱلۡمَيۡتَةَ وَٱلدَّمَ وَلَحۡمَ ٱلۡخِنزِيرِ وَمَآ
أُهِلَّ بِهِۦ لِغَيۡرِ ٱللَّهِۖ فَمَنِ ٱضۡطُرَّ غَيۡرَ
بَاغٖ وَلَا عَادٖ فَلَآ إِثۡمَ عَلَيۡهِۚ إِنَّ ٱللَّهَ
غَفُورٞ رَّحِيمٌ ١٧٣ إِنَّ ٱلَّذِينَ يَكۡتُمُونَ
مَآ أَنزَلَ ٱللَّهُ مِنَ ٱلۡكِتَٰبِ وَيَشۡتَرُونَ
بِهِۦ ثَمَنٗا قَلِيلًا ۙ أُوْلَٰٓئِكَ مَا يَأۡكُلُونَ
فِي بُطُونِهِمۡ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ ٱللَّهُ
يَوۡمَ ٱلۡقِيَٰمَةِ وَلَا يُزَكِّيهِمۡ وَلَهُمۡ عَذَابٌ
أَلِيمٌ ١٧٤ أُوْلَٰٓئِكَ ٱلَّذِينَ ٱشۡتَرَوُاْ
ٱلضَّلَٰلَةَ بِٱلۡهُدَىٰ وَٱلۡعَذَابَ بِٱلۡمَغۡفِرَةِۚ
فَمَآ أَصۡبَرَهُمۡ عَلَى ٱلنَّارِ ١٧٥ ذَٰلِكَ
بِأَنَّ ٱللَّهَ نَزَّلَ ٱلۡكِتَٰبَ بِٱلۡحَقِّۗ
وَإِنَّ ٱلَّذِينَ ٱخۡتَلَفُواْ فِي ٱلۡكِتَٰبِ لَفِي
شِقَاقِۭ بَعِيدٖ ١٧٦


يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُواْ كُلُواْ مِن
طَيِّبَٰتِ مَا رَزَقۡنَٰكُمۡ وَٱشۡكُرُواْ لِلَّهِ إِن
كُنتُمۡ إِيَّاهُ تَعۡبُدُونَ ١٧٢ إِنَّمَا حَرَّمَ
عَلَيۡكُمُ ٱلۡمَيۡتَةَ وَٱلدَّمَ وَلَحۡمَ ٱلۡخِنزِيرِ وَمَآ
أُهِلَّ بِهِۦ لِغَيۡرِ ٱللَّهِۖ فَمَنِ ٱضۡطُرَّ غَيۡرَ
بَاغٖ وَلَا عَادٖ فَلَآ إِثۡمَ عَلَيۡهِۚ إِنَّ ٱللَّهَ
غَفُورٞ رَّحِيمٌ ١٧٣ إِنَّ ٱلَّذِينَ يَكۡتُمُونَ
مَآ أَنزَلَ ٱللَّهُ مِنَ ٱلۡكِتَٰبِ وَيَشۡتَرُونَ
بِهِۦ ثَمَنٗا قَلِيلًا ۙ أُوْلَٰٓئِكَ مَا يَأۡكُلُونَ
فِي بُطُونِهِمۡ إِلَّا ٱلنَّارَ وَلَا يُكَلِّمُهُمُ ٱللَّهُ
يَوۡمَ ٱلۡقِيَٰمَةِ وَلَا يُزَكِّيهِمۡ وَلَهُمۡ عَذَابٌ
أَلِيمٌ ١٧٤ أُوْلَٰٓئِكَ ٱلَّذِينَ ٱشۡتَرَوُاْ
ٱلضَّلَٰلَةَ بِٱلۡهُدَىٰ وَٱلۡعَذَابَ بِٱلۡمَغۡفِرَةِۚ
فَمَآ أَصۡبَرَهُمۡ عَلَى ٱلنَّارِ ١٧٥
ذَٰلِكَ بِأَنَّ ٱللَّهَ نَزَّلَ ٱلۡكِتَٰبَ
بِٱلۡحَقِّۗ وَإِنَّ ٱلَّذِينَ ٱخۡتَلَفُواْ فِي
ٱلۡكِتَٰبِ لَفِي شِقَاقِۭ بَعِيدٖ ١٧٦

normal — no parfillskip

Figure 12: A narrower sample with no parfillskip (b).

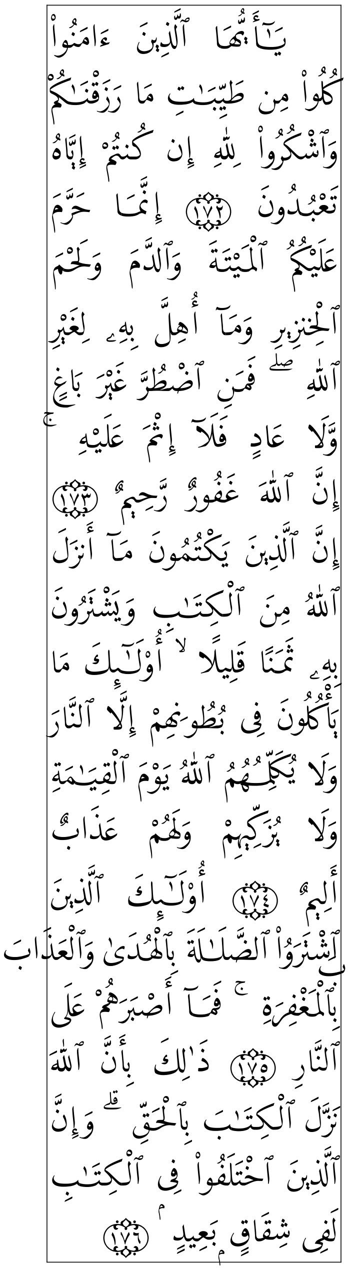


يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ كُلُوا۟ مِن
طَيِّبَٰتِ مَا رَزَقْنَٰكُمْ وَٱشْكُرُوا۟ لِلَّهِ
إِن كُنتُمْ إِيَّاهُ تَعْبُدُونَ ١٧٢ إِنَّمَا
حَرَّمَ عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ
ٱلْخِنزِيرِ وَمَآ أُهِلَّ بِهِۦ لِغَيْرِ ٱللَّهِ ۖ
فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ
فَلَآ إِثْمَ عَلَيْهِ ۚ إِنَّ ٱللَّهَ غَفُورٌ
رَّحِيمٌ ١٧٣ إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ
أَنزَلَ ٱللَّهُ مِنَ ٱلْكِتَٰبِ وَيَشْتَرُونَ
بِهِۦ ثَمَنًا قَلِيلًا ۙ أُو۟لَٰٓئِكَ مَا
يَأْكُلُونَ فِى بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا
يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ ٱلْقِيَٰمَةِ وَلَا
يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ أَلِيمٌ ١٧٤
أُو۟لَٰٓئِكَ ٱلَّذِينَ ٱشْتَرَوُا۟ ٱلضَّلَٰلَةَ
بِٱلْهُدَىٰ وَٱلْعَذَابَ بِٱلْمَغْفِرَةِ ۚ
فَمَآ أَصْبَرَهُمْ عَلَى ٱلنَّارِ ١٧٥
ذَٰلِكَ بِأَنَّ ٱللَّهَ نَزَّلَ ٱلْكِتَٰبَ
بِٱلْحَقِّ ۗ وَإِنَّ ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى
ٱلْكِتَٰبِ لَفِى شِقَاقٍۭ بَعِيدٍ ١٧٦

يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ كُلُوا۟ مِن
طَيِّبَٰتِ مَا رَزَقْنَٰكُمْ وَٱشْكُرُوا۟ لِلَّهِ
إِن كُنتُمْ إِيَّاهُ تَعْبُدُونَ ١٧٢ إِنَّمَا
حَرَّمَ عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ
ٱلْخِنزِيرِ وَمَآ أُهِلَّ بِهِۦ لِغَيْرِ ٱللَّهِ ۖ
فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ
فَلَآ إِثْمَ عَلَيْهِ ۚ إِنَّ ٱللَّهَ غَفُورٌ
رَّحِيمٌ ١٧٣ إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ
أَنزَلَ ٱللَّهُ مِنَ ٱلْكِتَٰبِ وَيَشْتَرُونَ
بِهِۦ ثَمَنًا قَلِيلًا ۙ أُو۟لَٰٓئِكَ مَا
يَأْكُلُونَ فِى بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا
يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ ٱلْقِيَٰمَةِ وَلَا
يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ أَلِيمٌ ١٧٤
أُو۟لَٰٓئِكَ ٱلَّذِينَ ٱشْتَرَوُا۟ ٱلضَّلَٰلَةَ
بِٱلْهُدَىٰ وَٱلْعَذَابَ بِٱلْمَغْفِرَةِ ۚ
فَمَآ أَصْبَرَهُمْ عَلَى ٱلنَّارِ ١٧٥
ذَٰلِكَ بِأَنَّ ٱللَّهَ نَزَّلَ ٱلْكِتَٰبَ
بِٱلْحَقِّ ۗ وَإِنَّ ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى
ٱلْكِتَٰبِ لَفِى شِقَاقٍۭ بَعِيدٍ ١٧٦

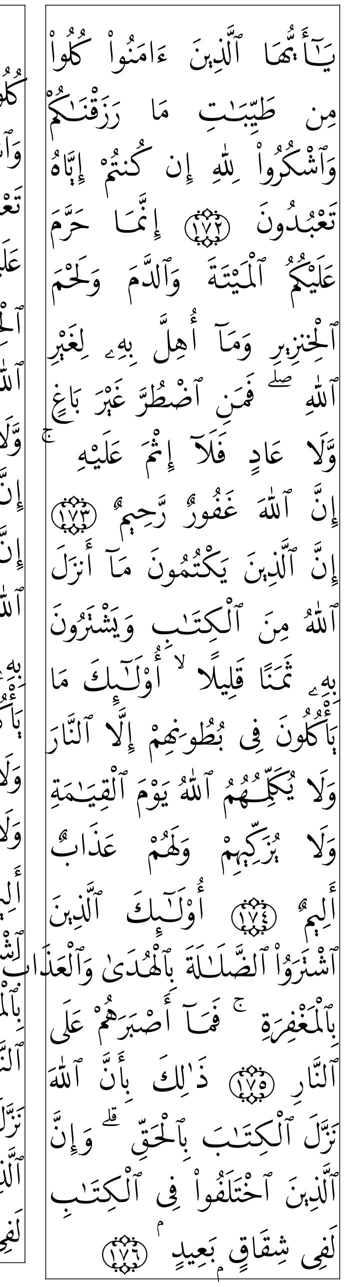
normal narrow

Figure 13: An even narrower sample (c).



يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟
كُلُوا۟ مِن طَيِّبَٰتِ مَا رَزَقْنَٰكُمْ
وَٱشْكُرُوا۟ لِلَّهِ إِن كُنتُمْ إِيَّاهُ
تَعْبُدُونَ ١٧٢ إِنَّمَا حَرَّمَ
عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ
ٱلْخِنزِيرِ وَمَآ أُهِلَّ بِهِۦ لِغَيْرِ
ٱللَّهِ ۖ فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ
وَلَا عَادٍ فَلَآ إِثْمَ عَلَيْهِ ۚ
إِنَّ ٱللَّهَ غَفُورٌ رَّحِيمٌ ١٧٣
إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ أَنزَلَ
ٱللَّهُ مِنَ ٱلْكِتَٰبِ وَيَشْتَرُونَ
بِهِۦ ثَمَنًا قَلِيلًا ۙ أُو۟لَٰٓئِكَ مَا
يَأْكُلُونَ فِى بُطُونِهِمْ إِلَّا ٱلنَّارَ
وَلَا يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ ٱلْقِيَٰمَةِ
وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ
أَلِيمٌ ١٧٤ أُو۟لَٰٓئِكَ ٱلَّذِينَ
ٱشْتَرَوُا۟ ٱلضَّلَٰلَةَ بِٱلْهُدَىٰ وَٱلْعَذَابَ
بِٱلْمَغْفِرَةِ ۚ فَمَآ أَصْبَرَهُمْ عَلَى
ٱلنَّارِ ١٧٥ ذَٰلِكَ بِأَنَّ ٱللَّهَ
نَزَّلَ ٱلْكِتَٰبَ بِٱلْحَقِّ ۗ وَإِنَّ
ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى ٱلْكِتَٰبِ
لَفِى شِقَاقٍۭ بَعِيدٍ ١٧٦

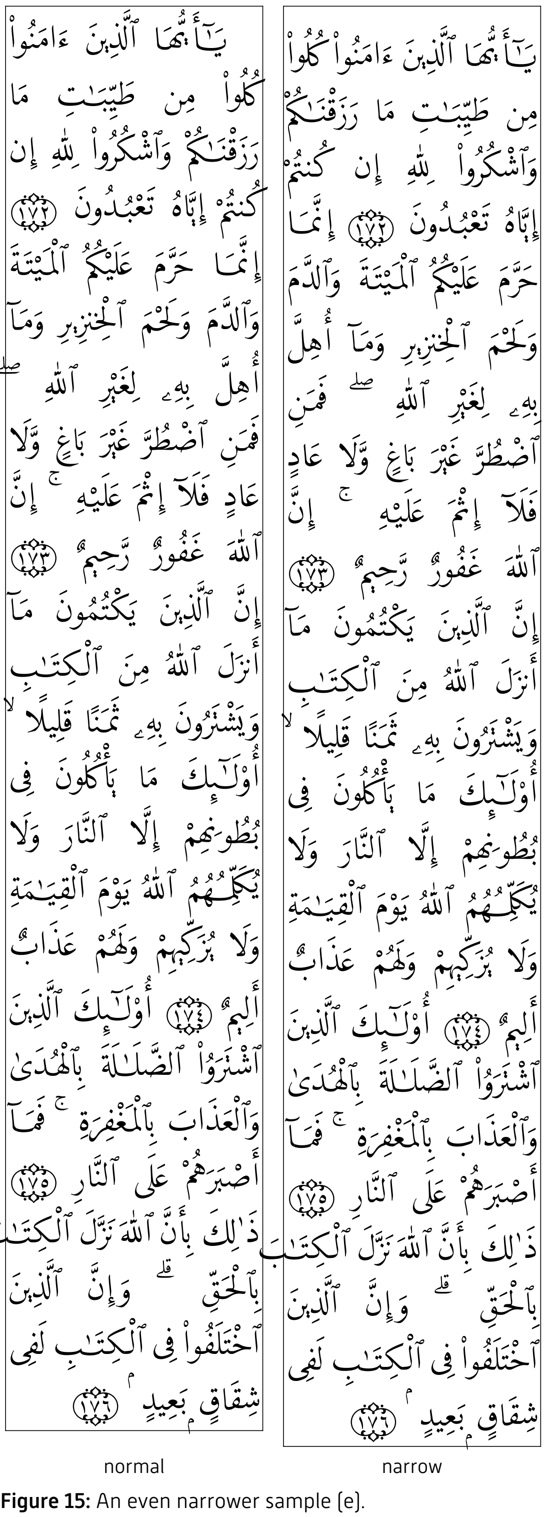
normal



يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ كُلُوا۟
مِن طَيِّبَٰتِ مَا رَزَقْنَٰكُمْ
وَٱشْكُرُوا۟ لِلَّهِ إِن كُنتُمْ إِيَّاهُ
تَعْبُدُونَ ١٧٢ إِنَّمَا حَرَّمَ
عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ وَلَحْمَ
ٱلْخِنزِيرِ وَمَآ أُهِلَّ بِهِۦ لِغَيْرِ
ٱللَّهِ ۖ فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ
وَلَا عَادٍ فَلَآ إِثْمَ عَلَيْهِ ۚ
إِنَّ ٱللَّهَ غَفُورٌ رَّحِيمٌ ١٧٣
إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ أَنزَلَ
ٱللَّهُ مِنَ ٱلْكِتَٰبِ وَيَشْتَرُونَ
بِهِۦ ثَمَنًا قَلِيلًا ۙ أُو۟لَٰٓئِكَ مَا
يَأْكُلُونَ فِى بُطُونِهِمْ إِلَّا ٱلنَّارَ
وَلَا يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ ٱلْقِيَٰمَةِ
وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ
أَلِيمٌ ١٧٤ أُو۟لَٰٓئِكَ ٱلَّذِينَ
ٱشْتَرَوُا۟ ٱلضَّلَٰلَةَ بِٱلْهُدَىٰ وَٱلْعَذَابَ
بِٱلْمَغْفِرَةِ ۚ فَمَآ أَصْبَرَهُمْ عَلَى
ٱلنَّارِ ١٧٥ ذَٰلِكَ بِأَنَّ ٱللَّهَ
نَزَّلَ ٱلْكِتَٰبَ بِٱلْحَقِّ ۗ وَإِنَّ
ٱلَّذِينَ ٱخْتَلَفُوا۟ فِى ٱلْكِتَٰبِ
لَفِى شِقَاقٍۭ بَعِيدٍ ١٧٦

narrow

Figure 14: An even narrower sample (d).



يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟
كُلُوا۟ مِن طَيِّبَٰتِ مَا
رَزَقْنَٰكُمْ وَٱشْكُرُوا۟ لِلَّهِ إِن
كُنتُمْ إِيَّاهُ تَعْبُدُونَ ﴿١٧٢﴾
إِنَّمَا حَرَّمَ عَلَيْكُمُ ٱلْمَيْتَةَ
وَٱلدَّمَ وَلَحْمَ ٱلْخِنزِيرِ وَمَآ
أُهِلَّ بِهِۦ لِغَيْرِ ٱللَّهِ ۖ
فَمَنِ ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا
عَادٍ فَلَآ إِثْمَ عَلَيْهِ ۚ إِنَّ
ٱللَّهَ غَفُورٌ رَّحِيمٌ ﴿١٧٣﴾
إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ
أَنزَلَ ٱللَّهُ مِنَ ٱلْكِتَٰبِ
وَيَشْتَرُونَ بِهِۦ ثَمَنًا قَلِيلًا ۙ
أُو۟لَٰٓئِكَ مَا يَأْكُلُونَ فِى
بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا
يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ ٱلْقِيَٰمَةِ
وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ
أَلِيمٌ ﴿١٧٤﴾ أُو۟لَٰٓئِكَ ٱلَّذِينَ
ٱشْتَرَوُا۟ ٱلضَّلَٰلَةَ بِٱلْهُدَىٰ
وَٱلْعَذَابَ بِٱلْمَغْفِرَةِ ۚ فَمَآ
أَصْبَرَهُمْ عَلَى ٱلنَّارِ ﴿١٧٥﴾
ذَٰلِكَ بِأَنَّ ٱللَّهَ نَزَّلَ ٱلْكِتَٰبَ
بِٱلْحَقِّ ۗ وَإِنَّ ٱلَّذِينَ
ٱخْتَلَفُوا۟ فِى ٱلْكِتَٰبِ لَفِى
شِقَاقٍۭ بَعِيدٍ ۘ ﴿١٧٦﴾

يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوا۟ كُلُوا۟
مِن طَيِّبَٰتِ مَا رَزَقْنَٰكُمْ
وَٱشْكُرُوا۟ لِلَّهِ إِن كُنتُمْ
إِيَّاهُ تَعْبُدُونَ ﴿١٧٢﴾ إِنَّمَا
حَرَّمَ عَلَيْكُمُ ٱلْمَيْتَةَ وَٱلدَّمَ
وَلَحْمَ ٱلْخِنزِيرِ وَمَآ أُهِلَّ
بِهِۦ لِغَيْرِ ٱللَّهِ ۖ فَمَنِ
ٱضْطُرَّ غَيْرَ بَاغٍ وَلَا عَادٍ
فَلَآ إِثْمَ عَلَيْهِ ۚ إِنَّ
ٱللَّهَ غَفُورٌ رَّحِيمٌ ﴿١٧٣﴾
إِنَّ ٱلَّذِينَ يَكْتُمُونَ مَآ
أَنزَلَ ٱللَّهُ مِنَ ٱلْكِتَٰبِ
وَيَشْتَرُونَ بِهِۦ ثَمَنًا قَلِيلًا ۙ
أُو۟لَٰٓئِكَ مَا يَأْكُلُونَ فِى
بُطُونِهِمْ إِلَّا ٱلنَّارَ وَلَا
يُكَلِّمُهُمُ ٱللَّهُ يَوْمَ ٱلْقِيَٰمَةِ
وَلَا يُزَكِّيهِمْ وَلَهُمْ عَذَابٌ
أَلِيمٌ ﴿١٧٤﴾ أُو۟لَٰٓئِكَ ٱلَّذِينَ
ٱشْتَرَوُا۟ ٱلضَّلَٰلَةَ بِٱلْهُدَىٰ
وَٱلْعَذَابَ بِٱلْمَغْفِرَةِ ۚ فَمَآ
أَصْبَرَهُمْ عَلَى ٱلنَّارِ ﴿١٧٥﴾
ذَٰلِكَ بِأَنَّ ٱللَّهَ نَزَّلَ ٱلْكِتَٰبَ
بِٱلْحَقِّ ۗ وَإِنَّ ٱلَّذِينَ
ٱخْتَلَفُوا۟ فِى ٱلْكِتَٰبِ لَفِى
شِقَاقٍۭ بَعِيدٍ ۘ ﴿١٧٦﴾

Figure 15: An even narrower sample (e).

Without going into a detailed analysis at the moment, we restrict ourselves to two critical observations.
First, in our tests one will notice that the glyph substitutions tend to take place on the right side of the line. They should be more evenly distributed throughout each line.
Second, we can say that the current method works better for alternate-shaped glyph substitution than it does for naturally-widened glyph substitution. This leads us to the next step in this research project:
Within the Husayni font there is now a mapping between flat extending via *tawīl* and curved widening via alternate glyphs. Consider the following manually typed utf text using the *tawīl* character (U+0640):

```
\ARROW\
```

In flat-extended typography that comes out like this:

فعيل ← فعيـــل

Husayni, through the optional Stylistic Alternates feature (`salt`) will map the flat *tawīl*-extended characters to curved widened characters. So with `salt=yes` selected in ConTEXt we get

فعيل ← فعيـــل

This opens up a way to connect a forthcoming solution to the flat *tawīl*-extended character method with the curved widened-glyph method. A future version of the optimizer may be able to optimize the paragraph in terms of the *tawīl* character and a set of rules along the lines we discussed earlier. Then we can simply convert the result to curves using the *tawīl* character. At least this is one possibility.
In any case, the current paragraph optimizer, even in its experimental status at the moment, represents one of the greatest and most important steps in the evolution of digital Arabic-script typography. Its potential impact on for Arabic-script typesetting is immense, and we excitedly look forward to its completion.

MlbibTEX and Its New Extensions

Jean-Michel Hufflen

These last years, MlbibTEX's kernel functions have been reused and extended in order to put new programs about bibliographies into action. Examples are the `hal` program, allowing an open archive site to be populated, the `mlbiblatex` program, building bibliographies suitable for the biblatex package, the `mlbibcontext` program, doing the same task for ConTEXt documents. We show how all these programs are organised, and explain how some operations can be refined or extended. For a point of view related to efficiency, the programs `mlbiblatex` and `mlbibcontext` are written using Scheme only, so they are more efficient than analogous programs that would interpret a .bst bibliography style of bibTEX.

I dedicate this article to my late father (1922–2012). When I was a child, he introduced me to the joy of reading. He was himself an avid reader; I surely share this feature with him.

1. Introduction

LATEX [23] is rightly viewed as a wonderful word processor for typesetting written documents. Besides, it is assisted by other programs like bibTEX [24] as bibliography processors which generate 'References' sections (.bbl files), or other graphical tools [4]. As a proof that TEX's community of developers is very dynamic, many programs—including LATEX itself—have evolved and been improved for many years. Other formats based on TEX or engines related to it have come out: e.g., XƎTEX [19], LuaTEX [7]. We can observe analogous dynamism about graphical tools: compare the two editions of *The LATEX Graphics Companion*, [5] and [4].

As we mentioned in [16], bibTEX was unrivalled as the bibliography processor usually associated with LATEX for a long time. Besides, bibTEX is stable for many years. In fact, some slight extensions, built out of bibTEX's source files, have been designed, e.g., bibTEX8 [23, § 13.1.1] and bibTEXu [29, § 4.3] (see [16] for more details). The difficulty of writing a new bibliography processor from scratch is mainly related to bibliography database files. Many LATEX users have a *huge* number of .bib files, according to the format used by bibTEX. So a new bibliography processor designed to work in conjunction with LATEX should be able to deal with this format. At first glance, it is not very complicated, entries' metadata are given using the syntax 'KEY = `value`', as you can see in Fig. 1. In reality, this format is more subtle. For example, values may be surrounded by double quotes:

```
TITLE = "Villa Vortex"
```

in which case a double quote character used within such a value must be surrounded by braces:

```
TITLE = "Die Energiej{\"a}ger"
```

Values may also be surrounded what braces[1]:

```
TITLE = {Grande Jonction}
```

in which case a double quote character can be used alone within such a value:

```
TITLE = {Murcos Verm\"achtnis}
```

The syntax for person names—see [10] for more details—is accurate for simple cases, but may be

[1] Personally, we always recommend users to adopt this convention, simpler, from our point of view.

```
@BOOK{holmstrom2011,
        AUTHOR = {Darwin Holmstrom},
        TITLE = {Toxic Terrain},
        SERIES = {Don Pendleton's The
                  Executioner},
        NUMBER = 390,
        PUBLISHER = {Gold Eagle},
        TOTALPAGES = 192,
        YEAR = 2011,
        MONTH = may}
```

Figure 1: Example using bibTEX's format.

surprising in such a case:

```
AUTHOR = {Jean {Le
Clerc de la Herverie}}
```

(if you remove the braces surrounding 'Le Clerc de la Herverie', that causes 'Herverie' to be viewed as the last name, 'Jean Le Clerc' as the first name, and 'de la' as a particle). In addition, many users get used to insert LATEX commands inside values of bibTEX fields:

```
TITLE = {\em Babylon Babies}
```

what would be difficult to interpret for a converter into a language used to put Web pages into action. Moreover, such a declaration:

```
TITLE = {\emph{Cosmos Incorporated}}
```

yields a title's specification which would be correctly interpreted by LATEX, but ConTEXt [6] would not recognise the `\emph` command.
In other words, it is quite easy to transform the syntax '`KEY = value`' into '`<KEY>value</KEY>`' if we adopt XML[2]-like syntax, or '`(KEY value)`' if Lisp[3]-like syntax is preferred. On the contrary, destructuring fields' values may be more complicated. That is why you can find many converters from .bib files into other formats, but at the first level. Roughly speaking, only a few programs run the risk of analysing the contents of fields' values.

```
<book id="holmstrom2011" from="mb.bib">
  <author>
   <name>
    <personname>
     <first>Darwin</first>
     <last>Holmstrom</last>
    </personname>
   </name>
  </author>
  <title>Toxic Terrain</title>
  <publisher>Gold Eagle</publisher>
  <number>390</number>
  <series>
   Don Pendleton's The Executioner
  </series>
  <totalpages>192</totalpages>
  <year>2011</year>
  <month><may/></month>
</book>
```

(The `from` attribute of the `book` element is set to the base name of the .bib file originally containing this entry.)
Figure 2: Fig. 1's example given using XML syntax.

Let us recall that we have developed MlbibTEX[4] [9], as a 'better' bibTEX with particular focus on multilingual features. As part of this task, we put into action an analysis of the values associated with bibTEX fields, as deeply as possible. We have precisely designed an internal format for bibliographical items. Later, we were asked for a program populating an open-archive site from the entries of .bib files [14,15].Although this program needed conventions more precise than usually about .bib files, we succeeded in developing it quickly. More precisely, they have many fragments in common, and the different parts were easily assembled. We decided to do again this kind of experiment… and succeeded again. First we explain how MlbibTEX can be extended. Second we recall some advantages of using MlbibTEX's kernel. Then we sketch the variants of MlbibTEX out.

[2] e**X**tensible **M**arkup **L**angage.
[3] **LIS**t **P**rocessor.
[4] **M**ulti**L**ingual bibTEX.

2. MlbibTEX's extensibility

When MlbibTEX's parser processes a .bib file, we can consider that it builds an XML tree of this file. More precisely, this program written using Scheme [18] builds expressions according to the SXML[5] format [20]. For example, Fig. 1's entry is translated to the XML tree given in Fig. 2. We can see that the author's name has been split into these components. Likewise, LATEX commands—e.g., `\em` or `\emph`—are recognised and replaced by XML tags.

When bibTEX users begin to run MlbibTEX, the most surprising feature is that the latter performs a more precise analysis of .bib files. When a field name is not recognised, a warning message is emitted[6]. By default, the fields subject to additional check are:

- the standard fields AUTHOR, EDITOR, MONTH, PAGES, and YEAR;
- the field DAY, used by numerous styles[7];
- the fields GENDER and TOTALPAGES, used by the bibliography styles associated with the jurabib package [23, § 12.5.1];
- two special fields used by MlbibTEX: LANGUAGE [9] and LASTSORTKEY [12].

The second extension of MlbibTEX—as abovementioned, the `hal` program, populating an open-archive site from the entries of .bib files [14]—needs additional check about the ADDRESS field of an entry being type @INPROCEEDINGS: we have to extract the country of the corresponding conference, and optionally the town. In addition, the name of such a country is to be checked, because we have to give its ISO[8] code. So we have decided to accept declarations like:

```
ADDRESS = {Breskens, The Netherlands}
```

or '`ADDRESS = {The Netherlands}`'. If the country is not given—e.g., in '`ADDRESS = {New-York}`' or:

```
ADDRESS = {Paris, Texas}
```

—an error has to be reported[9]. So we implemented a switch mechanism that allowed us to perform 'classical' check about this ADDRESS field when 'original' MlbibTEX was running, and 'complete' check when this program related to open archives was used[10]. Symmetrically, disabling some check procedures would be possible within other variants. When MlbibTEX's functions work in interpreted mode, such switch can be controlled by means of Scheme functions.

Later, we noticed the *modus operandi* of the biblatex package [21]: .bbl files only contain *structures*, and formatting 'References' sections is entirely deferred to LATEX. That is why there is no need of a `\bibliographystyle` command. If bibTEX is used, there is only one suitable bibliography style written using bibTEX's language.

[5] **S**cheme implementation of **XML**.

[6] This is just a warning message; the corresponding information is not lost. This *modus operandi* may be viewed as an advantage: for example, if you inadvertently type '`EDITORS = ...`' instead of '`EDITOR = ...`' inside an entry of type @INPROCEEDINGS, MlbibTEX will warn you whereas bibTEX will silently ignore that field. This feature may also be viewed as a drawback: if you specify a MONTH field, the associated value must be a symbol among `jan`, `feb`, ..., `dec`. Otherwise, MlbibTEX stops with an error message. This convention may appear as too restrictive, but MlbibTEX can sort w.r.t. month names, whereas bibTEX does not. To perform such an operation, month names must be recognised. Likewise, when years are to be sorted, MlbibTEX applies a numerical sort whereas bibTEX sorts years as strings, so the value associated with a YEAR field must be an integer.

[7] For example, the styles 'apa...', used by the American Psychology Association.

[8] **I**nternational **S**tandardisation **O**rganisation.

[9] We also accept declarations like: `ADDRESS = {Washington, District of Columbia, United States}` that is, a string of three comma-separated components. The first is supposed to be the town, the last the city.

[10] Technically, it is not very difficult since we consider that Scheme—as a functional programming language—allows functions to be handled like any other value. MlbibTEX's parser uses *association lists* whose elements look like `(key . f)` where f is the function to be called to parse the value associated with key. To perform such a switch, just change the function associated with key.

Another bibliography processor, biber [1], has come out: it builds only .bbl files suitable for biblatex. Let us consider the example of a LATEX document using this biblatex package given in Fig. 3. The corresponding .bbl file looks like Fig. 4, and the bibliography will be formatted w.r.t. the author-date style [23, § 12.3], because of the bibstyle option of the biblatex package.

```
\documentclass{article}

\usepackage[bibstyle=authoryear]{biblatex}
\addbibresource{mb.bib}  %  The suffix is
needed.

\begin{document}

Did you read \citetitle*{holmstrom2011}?
This is a
thriller written by \citeauthor{holmstrom2011}.

\printbibliography

\end{document}
```

Figure 3: Using the biblatex package.

```
\entry{holmstrom2011}{book}{}
  \name{author}{1}{}{%
    {{uniquename=0}{Holmstrom}{H.}{Darwin}%
     {D.}{}{}{}{}}%
  }%
  \field{title}{Toxic Terrain}%
  \list{publisher}{1}{{Gold Eagle}}%
  \field{number}{390}%
  \field{series}{Don Pendleton's The
Executioner}%
  \field{totalpages}{192}%
  \field{year}{2011}%
  \field{month}{05}%
\endentry
```

Figure 4: Reference used by the biblatex package.

The biblatex package's conceptor introduced new entry types a bibliography processor should be able to process. On the contrary, these new types are unknown in standard bibliography styles. Again, a switch mechanism allows us to recognise these new types only when the parser is running in a kind of '`mlbiblatex` mode'. Another point is related to *dates*: in standard bibliography styles, they are specified by a YEAR field and optionally by a MONTH field. The biblatex package allows dates to be expressed this way, or by means of a DATE field allowing the specification of a *range* of dates [21, § 2.3.8]. The extension of our parser for biblatex has been revised to include these points. Let us mention that the specification of dates are crucial within bibliographies since they are used for the sort operation in most styles. A last point: the syntax of the PAGES field has been refined.

A framework similar to biblatex had been put into action by Taco Hoekwater's `bib` module of ConTEXt [8]: see Fig. 5 for a source text using a bibliographical reference. This reference, as it should be produced by a bibliography processor, is given in Fig. 6. The `bib` module can be used with ConTEXt MkII [2], it has been reimplemented in ConTEXt MkIV by Hans Hagen [3]. In this last case, the switch we installed considers a new @CONTEXTPREAMBLE directive when a .bib file is parsed. This directive aims to replace the 'traditional' @PREAMBLE directive, often used to put definitions of new LATEX commands [23, § 13.2.4]. This @CONTEXTPREAMBLE directive can be used to program some LATEX commands put throughout .bib files and non-existing in ConTEXt.

```
\usemodule[bib] % Needed for MkII, not for
                % MkIV

\setupbibtex[database=mb]
\setuppublications[numbering=yes]

\starttext

Did you read \cite[holmstrom2011]?

\placepublications

\stoptext
```

Figure 5: Citations and bibliographies in ConTEXt.

```
\startpublication[k=holmstrom2011,
 t=book,a={{Holmstrom}},y=2011,n=2,s=Hol11]
\author[]{Darwin}[D.]{}{Holmstrom}
\pubyear{2011}
\title{Toxic Terrain}
\series{Don Pendleton’s The Executioner}
\volume{390}
\pubname{Gold Eagle}
\month{5}
\stoppublication
```

Figure 6: Reference used by ConTEXt.

3. MlbibTEX's advantages

When the approach of biblatex and ConTEXt is used, a bibliography processor does not have to provide the text of successive references of a bibliography. Since it just produces structures whatever the bibliography style is—such a style is put into action by customising the command of LATEX or ConTEXt producing the final bibliography—the idea is to build two accurate bibliography processors out of MlbibTEX's kernel. These two programs —`mlbiblatex` (resp. `mlbibcontext`) for biblatex (resp. ConTEXt)—are written entirely in Scheme, in order to get more efficiency. Even if we are not interested in multilingual extensions of MlbibTEX during a first step, here are the features of interest for such bibliography processors.

3.1 Order relations

In [11], we showed how the lexicographic order relations handled by MlbibTEX were built. These order relations—implemented by means of Scheme functions—are language-dependent. A simple use of the `<english?` function—for English words—to compare two strings is given by the first example of Fig. 7—'`#t`' (resp. '`#f`') stands for the 'true' (resp. 'false') value in Scheme. In reality, these functions are more powerful since they use optional arguments—controlling the behaviour—in addition to the two strings to be compared:

- the third is a *thunk*[11] that is called if the two strings are equal;
- the fourth is < (resp. >) for an ascending (resp. a descending) order;
- the fifth is `#f` for a case-insensitive comparison, `uppercase-1st` (resp. `lowercase-1st`) if uppercase (resp. lowercase) letters take precedence when two strings are different only by the case.

Fig. 7's second example shows the default values of these three additional arguments. By default, these functions implement *strict* order relations, that is, *irreflexive*, asymmetric, and transitive; as < for numbers. The sixth example shows that our `<english?` function defaults to a case-sensitive relation in which uppercase letters take precedence over lowercase ones, the seventh example shows how to proceed if you would like lowercase letters to take precedence. Finally, the last example shows how the third argument can be used to *chain* order relations[12]: the idea is to sort persons regarding last names, first names, birth dates, and possibly other information. As you can see, this feature—sketched in [12, § 4]—makes easier a sort by means of several successive sort keys. More details about these order relations are given in [17].

3.2 Syntactical extensions

MlbibTEX's syntactical extensions about multilinguism are explained in detail in [9]. Presently, they are not used by the programs `mlbiblatex` and `mlbibcontext`. On the contrary, our extensions for authors' and editors' names can be directly usable by these two programs. In addition to bibTEX's conventions, *keywords* may be used to point to the four parts—*First*, *von*, *Last*, *Junior*—of a name, what may be very useful:

[11] A zero-argument function, w.r.t. Scheme's terminology.

[12] The `arithmetical?` function, used within Fig. 7's last example is analogous to our order relations, in the sense that its third argument is called if the two numbers given as first two arguments are equal. Otherwise it behaves like <.

```
(<english? "ConTeXt" "ConTeXt")                                  ⟹  #f
(<english? "ConTeXt" "ConTeXt" (lambda () #f) < 'uppercase-1st)  ⟹  #f   ; Default values explicited.
(<english? "ConTeXt" "ConTeXt" (lambda () 'ok))                  ⟹  ok   ; Equal strings.
(<english? "ConTeSt" "ConTeXt")                                  ⟹  #t
(<english? "ConTeSt" "ConTeXt" (lambda () 'ok) >)                ⟹  #f   ; Descending order.
(<english? "ConText" "ConTeXt" (lambda () 'ok))                  ⟹  #f
(<english? "ConText" "ConTeXt" (lambda () 'ok) < #f)             ⟹  ok   ; Case-insensitive equality.
(<english? "ConText" "ConTeXt" (lambda () 'ok) < 'lowercase-1st) ⟹  #t   ; Lowercase letters take
                                                                          ; precedence.
(<english? "ConTeXt" "ConTeSt"
           (lambda ()
              (<english? "Mk" "Mk" (lambda () (<arithmetical? 2 4 (lambda () ...))))) ⟹ #f
```

Figure 7: Order relations handled by MlbibTEX.

```
first => Jean, last =>
Le Clerc de la Herverie
```

[the four keywords '`first =>`', '`von =>`', '`last =>`', '`junior =>`' are available, the order of appearance being irrelevant]. In addition, the '`abbr =>`' keyword may be used when a first name is not abbreviated according to the standard way, that is, retaining only the first letter. If an organisation's name is used as an author or editor, you can use the keywords '`org =>`' for the name as it must be typeset and '`sortingkey =>`' for the key used for sorting:

```
org => Euro\TeX~2012,
sortingkey => EuroTeX 2012
```

It is well-known that co-authors are connected by means of the ' and ' keyword. MlbibTEX also allows the specification of *collaborators*, by means of the ' with ' keyword; an example is given in this article's bibliography: see the reference [23].

4. MlbibTEX's programs

MlbibTEX's distribution is located at:
http://disc.univ-fcomte.fr/home/~jmhufflen /texts/superreport/smlbibtex-1.3.tar.gz
The easiest way to install it is to compile the source files by the bigloo [25] Scheme compiler; the installation procedure [17] uses the commands `configure` [28] and `make` [22], well-known within GNU[13] software; more details are given in [17, § 4.2]. The executable programs generated are described hereafter. The complete distribution's version number is given 'classically', that is, by means of sequence of numbers. Versions of particular variants are labelled by geographical names. Those demonstrated at the EuroTEX 2012 conference are 'Breskens versions'.

4.1 `mlbibtex`

This programs aims to replace bibTEX and is described in [9]; you can use it analogously to 'original' bibTEX. This `mlbibtex` is the 'historical' origin of the present toolbox.

4.2 `mlbibtex2xml`

This program allows .bib files to be converted into XML files, according to the format internally used by MlbibTEX. You can run it as follows:

```
mlbibtex2xml ([-screen] | [-o output])\
    f0.bib f1.bib ...
```

where f_0`.bib`, f_1`.bib`, ...—the .bib suffix can be omitted—are .bib files. If the `-screen` option is used, the result is displayed at the screen, otherwise it is written into a file. If the `-o` option is used, `output` gives the output file name, oth-

[13] Recursive acronym: **G**nu is **N**ot **U**nix.

erwise, this name defaults to f_0-`mlbiblio.xml`, even if several .bib files are processed. Obviously, results look like Fig. 2.

4.3 `ar-style` and `hal`

These two programs are the first two extensions of MlbibTEX. The `ar-style` program can be used for activity reports' bibliographies, when they have to be conformant to the classification of the French agency AERES[14] [13]. See Section 'MlbibTEX's extensibility' and [14,15] about the `hal` program.

4.4 `mlbiblatex`

The `mlbiblatex` program builds .bbl files suitable for the biblatex package. You can run it as follows:

```
mlbiblatex filename.aux
    key-expr lg-code
```

where:

`filename.aux`
: —the `.aux` suffix can be omitted—is the auxiliary file where the information about bibliographical keys and database files has been stored;

`key-expr`
: gives successive sort keys, according to the pattern `(m | n | t | y)*`, where 'm', 'n', 't', 'y' respectively stand for '**M**onth'[15], '**N**ame' (person name as an author or editor), '**T**itle', '**Y**ear'; all the other signs are ignored; there is no default order relation[16]: if no sign is recognised, the list of bibliographical items is left unsorted[17];

`lg-code`
: is the code for the language to be used for sorting strings—this information is relevant whenever person names and titles of works are compared—available values are DE for German, EN for English, FR for French, PO for Polish; there is no default value.

Results look like Fig. 4. More detais are given in [17].

4.5 `mlbibcontext`

The `mlbibcontext` program builds .bbl files suitable for ConTEXt. The corresponding command line looks like `mlbiblatex`'s:

```
mlbibcontext filename.aux
    key-expr lg-code
```

and `filename.aux`, `key-expr`, `lg-code` have the same meaning. Results look like Fig. 6.

5. Future directions

As we mention above, the interface between the functions of a word processor in charge of processing 'References' sections—the commands of the biblatex package or ConTEXt MkIV—could be improved. For example, the commands `mlbiblatex` and `mlbibcontext` only deal with ascending orders. This is just related to the rough interface we designed in order to propose first experimental versions of these programs: as shown in Section 'MlbibTEX's advantages', descending orders are provided by MlbibTEX's kernel. Concerning the biblatex package, we think that an option could be added[18]:

```
\usepackage
    [backend=mlbiblatex,...]%
    {biblatex}
```

other options allowing accurate information to be passed to MlbibTEX.

[14] ***A****gence d'****É****valuation de la* ***R****echerche et de l'****E****nseignement* ***S****upérieur*, that is, 'agency evaluating research and university courses'.

[15] … an item without month information being ranked after an item with such.

[16] The default order relation used by both bibTEX and biber would be specified by `ynt`. Let us recall that by default, these two programs do not use any information about month during the sort step.

[17] In this case, the bibliography is *unsorted*, that is, the order of items is the order of first citations of these items throughout the document.

[18] Presently, the possible values for the backend option of biblatex are '`bibtex`', '`bibtex8`', '`bibtexu`', '`biber`'.

Likewise, ConTEXt MkIV users should be able to choose between bibTEX—or an 'enriched' bibTEX such that bibTEX8 or bibTEXu—or MlbibTEX. In this last case, we have to study how accurate information could be passed to the `mlbibcontext` program.

Some present lack of MlbibTEX: only two encodings are available, for input .bib files as well as output .bbl ones. More precisely, .bib files are supposed to be encoded w.r.t. Latin 1. The characters that are not included in this encoding—e.g., some Polish letters, such that 'ł'—can be reached only by using TEX commands—like '`\l`'[19]. About generated .bbl files, either MlbibTEX detects that the Latin 1 encoding is used by looking into the document's preamble[20], in which case this encoding is used for the .bbl file produced; otherwise, this .bbl file is a pure ASCII[21] file, all the accented letters being specified by means of TEX commands[22]. Such behaviour is due to the Scheme programming language. MlbibTEX has been written using the fifth revision of this language [18], not Unicode-compliant. Most of Scheme interpreters can deal with Latin 1, some—not all—accept other encodings, but in a non-portable way. Besides, we want our functions to be able to work on as many Scheme interpreters as possible. A new revision of Scheme is in progress[23] and will be Unicode-compliant, so a future version of MlbibTEX should be able to deal with other encodings such that Latin 2, UTF-8, UTF-16, etc.

Last but not at least, we plan to update the programs `mlbiblatex` and `mlbibcontext`, in order for them to be able to deal with MlbibTEX's multilingual features. From our point of view, that should be quite easy for `mlbibcontext`, in the sense that all the languages are available *a priori* within ConTEXt MkIV—you do not have to put all the languages you use throughout a text as options of a module like the babel package [23, Ch. 9]—but might require more work for the texts to be processed by the commands of the biblatex package.

6. Conclusion

We are personally an adept of functional programming in general and Scheme in particular. But MlbibTEX has been able to be adapted to applications other than those initially planned, what is a good quality for a program[24]. In particular, the `mlbiblatex` program succeded in taking as much advantage as possible of biblatex's features[25] with just slight modifications of our kernel. We think that we have been able to reach such adaptability and flexibility because of the use of Scheme, even if these qualities could

[19] For example, the name of the Polish city 'Łódź' should be written down '`{\L }\'{o}d\'{z}`' or '`{\L }`ód`\'{z}`' within a .bib file, its internal form handled by MlbibTEX is '`{\L }`ód`\'{z}`', since 'ó' belongs to Latin 1, whereas 'Ł' and 'ź' do not.

[20] bibTEX just read .aux files and never reads a .tex file [23, § 12.1.3], whereas the `mlbibtex` program may look into a document's preamble.

[21] **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange.

[22] Let us recall that ConTEXt MkIV texts are supposed to be encoded w.r.t. UTF-8. Since MlbibTEX cannot deal with this encoding, the output files of the `mlbibcontext` program are presently encoded w.r.t. pure ASCII.

[23] See the Web page http://scheme-reports.org. In fact, MlbibTEX has been implemented using the conventions of R5RS, what stands for '**R**evised^5 **R**eport on the algorithmic language **S**cheme' [18]. Later, a new revision (R6RS) was designed and ratified [26][27], including functions dealing with the whole range of Unicode and different encodings [27, §§ 1 & 2.9]—but for some reasons that we do not give here, most Scheme implementors did not update their programs. So MlbibTEX is still R5RS-compliant. It seems that Scheme's next version (R7RS)—see some drafts at the Web page abovementioned—will be adopted by most Scheme implementors. So we hope that we will be able to get a Unicode-compliant version of MlbibTEX very soon.

[24] More generally, some people already announced the end of Lisp dialects, or the end of TEX & Co… and these programs are still in action.

[25] Especially the notion of field *type*: for example, `@AUTHOR` is a *list of names*, `@TITLE` is a *literal*, according to the biblatex package's terminology. Analogous notions exist within MlbibTEX.

have been reached within other programming paradigms[26]. In addition, our programs can be used with a Scheme interpreter, but better efficiency is reached if programs are compiled. Even if we think that we are not in competition with a bibliography processor like biber, it is certain that a program written using Scheme is more efficient than a program written using Perl[27]. So we have spent much time when we began MlbibTEX's development, but we do not regret anything and were happy to be able to adapt this program to new requirements.

7. References

[1] François Charette and Philip Kime: *biber. A Backend Bibliography Processor for biblatex. Version biber 0.9 (biblatex 1.6)*. August 2011. http://freefr.dl.sourceforge.net/project/biblatex-biber/biblatex-biber/development/documentation/biber.pdf.

[2] CONTEXTGARDEN, http://wiki.contextgarden.net/Bibliography: *Bibliographies in MkII*. April 2012.

[3] CONTEXTGARDEN, http://wiki.contextgarden.net/Bibliography_mkiv: *Bibliographies in MkIV*. July 2012.

[4] Michel Goossens, Frank Mittelbach, Sebastian Rahtz, Denis B. Roegel and Herbert Voß: *The LATEX Graphics Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. January 2009.

[5] Michel Goossens, Sebastian Rahtz and Frank Mittelbach: *The LATEX Graphics Companion. Illustrating Documents with TEX and PostScript*. Addison-Wesley Publishing Company, Reading, Massachusetts. March 1997.

[6] Hans Hagen: *ConTEXt, the Manual*. November 2001. http://www.pragma-ade.com/general/manuals/cont-enp.pdf.

[7] Hans Hagen: 'The Luafication of TEX and ConTEXt'. In: *Proc. BachoTEX 2008 Conference*, p. 114–123. April 2008.

[8] Taco Hoekwater: 'The Bibliographic Module for ConTEXt'. In: *EuroTEX 2001*, p. 61–73. Kerkrade (the Netherlands). September 2001.

[9] Jean-Michel Hufflen: 'MlbibTEX's Version 1.3'. TUG*boat*, Vol. 24, no. 2, p. 249–262. July 2003.

[10] Jean-Michel Hufflen: 'Names in bibTEX and MlbibTEX'. TUG*boat*, Vol. 27, no. 2, p. 243–253. TUG 2006 proceedings, Marrakesh, Morocco. November 2006.

[11] Jean-Michel Hufflen: 'Managing Order Relations in MlbibTEX'. TUG*boat*, Vol. 29, no. 1, p. 101–108. EuroBachoTEX 2007 proceedings. 2007.

[12] Jean-Michel Hufflen: 'Revisiting Lexicographic Order Relations on Person Names'. In: *Proc. BachoTEX 2008 Conference*, p. 82–90. April 2008.

[13] Jean-Michel Hufflen: *Classe superreport — Manuel d'utilisation*. March 2010. http://lifc.univ-fcomte.fr/home/~jmhufflen/superreport/superreport-readme.pdf.

[14] Jean-Michel Hufflen: 'Using MlbibTEX to Populate Open Archives'. In: Tomasz Przechlewski, Karl Berry, Gaby Gic-Grusza, Ewa Kolsar and Jerzy B. Ludwichowski, eds., *Typographers and Programmers: Mutual Inspirations. Proc. BachoTEX 2010 Conference*, p. 45–48. April 2010.

[15] Jean-Michel Hufflen: 'From Bibliography Files to Open Archives: the Sequel'. In: Karl Berry, Jerzy B. Ludwichowski and Tomasz Przechlewski, eds., *Proc. EuroBachoTEX 2011 Conference*, p. 61–66. Bachotek, Poland. April 2011.

[16] Jean-Michel Hufflen: 'A Comparative Study of Methods for Bibliographies'. TUG*boat*, Vol. 32, no. 3, p. 289–301. Proc. TUG 2011 conference. October 2011.

[17] Jean-Michel Hufflen: 'MlbibTEX and the biblatex package'. In: Tomasz Przechlewski, Karl Berry and Jerzy B. Ludwichowski, eds., *Twenty Years After. Proc. BachoTEX 2012 Conference*, p. 91–99. Bachotek, Poland. April 2012.

[18] Richard Kelsey, William D. Clinger, and Jonathan A. Rees, with Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman and Mitchell Wand: 'Revised^5 Report on the Algorithmic Language Scheme'. HOSC, Vol. 11, no. 1, p. 7–105. August 1998.

[19] Jonathan Kew: 'XƎTEX in TEX Live and beyond'. TUG*boat*, Vol. 29, no. 1, p. 146–150. EuroBachoTEX 2007 proceedings. 2007.

[20] Oleg E. Kiselyov: XML *and Scheme*. September 2005. http://okmij.org/ftp/Scheme/xml.html.

[21] Philipp Lehman: *The biblatex Package. Programmable Bibliographies and Citations. Version 1.6*. 29

[26] But we think that more effort would have been needed.

[27] **P**ractial **E**xtraction and **R**eport **L**anguage. A good introduction to this language is [30].

July 2011. ftp://ftp.tex.ac.uk/archive/Archive\%20directory/macros/latex/exptl/biblatex/doc/biblatex.pdf.

[22] Miki Loukides and Andy Oram: *Programming with* GNU *Software*. O'Reilly & Associates, Inc. December 1996.

[23] Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig and Joachim Schrod: *The LATEX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.

[24] Oren Patashnik: *bibTEXing*. February 1988. Part of the bibTEX distribution.

[25] Manuel Serrano: *Bigloo. A Practical Scheme Compiler. User Manual for Version 3.3b*. March 2010.

[26] Michael Sperber, R. Kent Dybvig, Matthew Flatt, and Anton van Straaten, with Richard Kelsey, William Clinger, Jonathan Rees, Robert Bruce Findler and Jacob Matthews: *Revised^6 Report on the Algorithmic Language Scheme*. September 2007. hhtp://www.r6rs.org.

[27] Michael Sperber, R. Kent Dybvig, Matthew Flatt, and Anton van Straaten, with Richard Kelsey, William Clinger and Jonathan Rees: *Revised^6 Report on the Algorithmic Language Scheme—Standard Libraries*. September 2007. hhtp://www.r6rs.org.

[28] Gary V. Vaughn, Ben Ellison, Tom Tromey and Ian Lance Taylor: GNU *Autoconf, Automake, and Libtool*. Sams. October 2000.

[29] Herbert Voß: *Bibliografien mit LATEX*. Lehmans Media, Berlin. 2011.

[30] Larry Wall, Tom Christiansen and Jon Orwant: *Programming Perl*. 3rd edition. O'Reilly & Associates, Inc. July 2000.

Demonstration of the ‘mlbibcontext’ Program

Jean-Michel Hufflen

This short statement aims to sketch the broad outlines of the presentation performed at the 6th ConTEXt meeting.

Introduction

When the bibTEX bibliography processor [24] builds a ‘Reference’ section for a source text typeset by the LATEX word processor [23], it only uses information stored in auxiliary (.aux) files [23, § 12.1.3]. In particular, such an .aux file gives the *bibliography style* to be used, as a .bst file[1]. Such a style is monolithic, in the sense that nothing can be customised when bibTEX is called: for example, the order relation used to sort bibliographical items is hard-wired in any .bst file. The biber program [1]—often used in cojunction with the biblatex package [21]—is more flexible: when it runs, it uses a configuration file (.bcf[2]) file—using XML[3]-like syntax—as explained in [16, § 2.5]: in particular, such a .bcf file allows the sort of bibliographical items to be customised. However, let us recall that biber has a drawback from a point of view related to ConTEXt: it only builds ‘References’ sections suitable for the biblatex package. As explained in [10], the `mlbibcontext` program aims to build ‘References’ sections suitable for the bibliography support for ConTEXt [2,3]. The main point of the demonstration is to show which information is needed by mlbibcontext, in order for this program to be as powerful as possible. In other words, we aim to help design a nice interface between ConTEXt and `mlbibcontext`[4].

Plan

Let us recall that the `mlbibcontext` program—written entirely using the Scheme programming language [18]—builds ‘References’ sections suitable for the commands of Taco Hoekwater’s `bib` module [8], reimplemented within ConTEXt MkIV by Hans Hagen [3]. The demonstration will focus on the following points:

- its installation: the easiest way is to compile the source files by the bigloo [25] Scheme compiler[5]; the installation procedure [17] uses the commands `configure` [28] and `make` [22], well-known within GNU[6] software; the source files are available at the Web page http://disc.univ-fcomte.fr/home/~jmhufflen/texts/superreport/smlbibtex-1.3.tar.gz;
- the `mlbibcontext` program allows order relations used to sort bibliographies to be customised w.r.t. successive keys given by bibTEX’s fields [10,11]; only ascending orders can be used presently,

[1] Except if the biblatex package is used [21], in which case the bibliography style applied by bibTEX is implicitly the biblatex bibliography style.

[2] **B**iber **C**onfiguration **F**ile.

[3] e**X**tensible **M**arkup **L**anguage.

[4] Let us mention that `mlbibcontext` could deal with configurations described by XML files—in particular, it could process bibliographical entries given using XML-like syntax—; it can also process additional definitions written using the Scheme programming language [18].

[5] Of course, it is preferable for `mlbibcontext` to be compiled, in order to get more efficiency. The use of other Scheme compilers or interpreters is possible.

[6] Recursive acronym: **G**nu is **N**ot **U**nix.

but this point could be improved by a nicer interface: the kernel of MlbibTEX[7] also provides descending order relations;

- the `mlbibcontext` program allows you to put many basic commands of LATEX inside values of bibTEX's fields, even if the result is processed by ConTEXt; moreover, some commands specific to ConTEXt may be grouped into a special preamble within .bib files: the `@CONTEXTPREAMBLE` directive instead of the traditional `@PREAMBLE` directive [6].

To end up, let us mention the `mlbibtex2xml` program [10], part of MlbibTEX. This program allows bibliographical items to be given using XML-like syntax. This kind of text can be processed by ConTEXt MkIV (cf. [7, Fig. 8]). However, we think that `mlbibtex2xml`'s outputs could be processed by programs written using Lua [12]—as allowed by ConTEXt MkIV [7]—rather than ConTEXt' features related to TEX. When .bib files are processed by `mlbibtex2xml`, no sort operation is performed.

References

[1] François Charette and Philip Kime: *biber. A Backend Bibliography Processor for biblatex. Version biber 0.9 (biblatex 1.6)*. August 2011. http://freefr.dl.sourceforge.net/project/biblatex-biber/biblatex-biber/development/documentation/biber.pdf.

[2] CONTEXTGARDEN, http://wiki.contextgarden.net/Bibliography: *Bibliographies in MkII*. April 2012.

[3] CONTEXTGARDEN, http://wiki.contextgarden.net/Bibliography_mkiv: *Bibliographies in MkIV*. July 2012.

[4] Hans Hagen: 'The Luafication of TEX and ConTEXt'. In: *Proc. BachoTEX 2008 Conference*, p. 114–123. April 2008.

[5] Taco Hoekwater: 'The Bibliographic Module for ConTEXt'. In: *EuroTEX 2001*, p. 61–73. Kerkrade (the Netherlands). September 2001.

[6] Jean-Michel Hufflen: 'MlbibTEX Meets ConTEXt'. TUG*boat*, Vol. 27, no. 1, p. 76–82. EuroTEX 2006 proceedings, Debrecen, Hungary. July 2006.

[7] Jean-Michel Hufflen: 'Processing 'Computed' Texts'. MAPS, Vol. 41, p. 68–78. 2010.

[8] Jean-Michel Hufflen: 'A Comparative Study of Methods for Bibliographies'. TUG*boat*, Vol. 32, no. 3, p. 289–301. Proc. TUG 2011 conference. October 2011.

[9] Jean-Michel Hufflen: 'MlbibTEX and the biblatex package'. In: Tomasz Przechlewski, Karl Berry and Jerzy B. Ludwichowski, eds., *Twenty Years After. Proc. BachoTEX 2012 Conference*, p. 91–99. Bachotek, Poland. April 2012.

[10] Jean-Michel Hufflen: 'MlbibTEX and Its New Extensions'. In: *Proc. EuroTEX 2012*. Breskens, The Netherlands. October 2012.

[11] Jean-Michel Hufflen: *Gestion d'ordres lexicographiques multilingues avec xindy et MlbibTEX*. À paraître dans les *Cahiers GUTenberg*. 2012.

[12] Roberto Ierusalimschy: *Programming in Lua*. 2nd edition. Lua.org. March 2006.

[13] Richard Kelsey, William D. Clinger, Jonathan A. Rees, Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman and Mitchell Wand: 'Revised^5 Report on the Algorithmic Language Scheme'. HOSC, Vol. 11, no. 1, p. 7–105. August 1998.

[14] Philipp Lehman: *The biblatex Package. Programmable Bibliographies and Citations. Version 1.6*. 29 July 2011. ftp://ftp.tex.ac.uk/archive/Archive%20directory/macros/latex/exptl/biblatex/doc/biblatex.pdf.

[15] Miki Loukides and Andy Oram: *Programming with* GNU *Software*. O'Reilly & Associates, Inc. December 1996.

[16] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig and Joachim Schrod: *The LATEX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.

[17] Oren Patashnik: *bibTEXing*. February 1988. Part of the bibTEX distribution.

[18] Manuel Serrano: *Bigloo. A Practical Scheme Compiler. User Manual for Version 3.3b*. March 2010.

[19] Gary V. Vaughn, Ben Ellison, Tom Tromey and Ian Lance Taylor: GNU *Autoconf, Automake, and Libtool*. Sams. October 2000.

[7] **M**ulti**L**ingual bibTEX.

Abstracts without papers

Run for Fun

Jano Kula

Sports and especially long distance runs are known for the good doses of endorphin. Instead, we will show some adrenaline challenges while preparing such a sport event: 10 km run through the historic center of Prague. Plotters, tables, layers, composition.

ConTEXt: the script

Hans Hagen

The ConTEXt runner `context` has inherited a few features from its predecessor `texexec`. Instead of hardcoding functionality for typesetting listings and manipulating PDF files in the script itself they are now isolated in TEX files. In this presentation I will show some of these lesser known features of the script.

ConTEXt: after the cleanup

Hans Hagen

After the transition from MkII to MkIV a cleanup stage has been started. What is involved in this cleanup and what will happen afterwards. This is more a discussion than a presentation and users are invited to express their wishes and priorities.

Metapost workshop

Mari Voipio

'A pragmatic approach to MetaPost', or, 'How to get useful results out of MetaPost if are not a programmer, are not a mathematician, and are a complete beginner besides.'

A couple of styles

Hans Hagen

When you keep an eye on what modules get added to ConTEXt, you will notice that quite some of them are a mixture of TEX, METAPOST and Lua. I will show a few that might have gone unnoticed. They can often serve as an example for your own local usage.

Lexing

Hans Hagen

As I use SciTE most of the time, there is some mutual influence between coding and visualized in this editor. Especially the possibility to write more advanced lexers has lead to some changes (for the good) in the code base. Here I will show some of that (as it might help those who browse the source).

(visual) debugging

Hans Hagen

Compared to MkII the MkIV code has more tracing on board. At the Lua end we have trackers and recently a start has been made to extend that to the TEX end, where it will replace the `\trace*true` like macros. As part of the cleanup the original visual debugger module has been replaced by an even less intrusive variant. It provides the usual visual clues about what goes on the page. The new mechanism is more advanced that the old one but still assumes some knowledge of what happens inside TEX. In this presentation we will explain some of this.

Japanese Typesetting with LuaTEX

KITAGAWA Hironori

There are some issues for typeset Japanese documents by LuaTEX. Some of them, such as end-of-line rule and the value of a grouped variable 'at the end of a `\hbox`', are (partially) resolved by writing Lua codes. Also we can discuss the specification of LuaTEX on vertical typesetting, referring to that of Japanese pTEX.

Mixed columns

Hans Hagen

One of the last things to redo in MkIV is the page builder. Although bits and pieces have been redone, some major effort is needed to upgrade multi columns mechanisms. We (currently) have three mechanisms: regular columns that can be mixed with single column mode, simple columns that can be used in a boxed way, and columnsets. The first two have been replaced by a new mechanism tagged as mixed columns. This mechanism permits instances of multicolumns, either or not in the page flow or in boxes, and the old mechanisms will go. Of course we try to remain compatible as much as possible. In this talk we can discuss some of the issues involved and identify future needs.

Differences in typesetting rules between Czech and Slovak languages (in the context of ConTEXt)

Tomás Hála

During the existence of Czechoslovakia, Czech and Slovak typesetting rules were defined by one common norm. At present, Slovak rules have mostly been fixed by official documents whereas Czech rules are rather custom based.

This contribution deals with comparison of the rules in both languages, especially with the use of hyphen, dashes, lists etc. In addition to that, some ot these items in Czech and Slovak differ considerably from those in other languages. All important items have been compared with facilities in the typesetting system ConTEXt and it seems that some situations have not been covered in configuration files. Therefore several suggestions for language settings have been made in order to make ConTEXt more general and comfortable for ordinary users.

Participant list of the 6th ConTEXt meeting

Leo Arnold, Technische Universität München,
Garching bei München, Germany
latex@arney.de

Doris Behrendt, Gymnasium Marktbreit,
Biebelried, Germany
doris.behrendt@me.com

Sietse Brouwer,
The Netherlands
sbbrouwer@gmail.com

Györgyi Bujdosó, Faculty of Computer Science,
University of Debrecen,
Debrecen, Hungary
bujdoso.gyongyi@inf.unideb.hu

Andreas Dafferner, Heidelberger Akademie der Wissenschaften,
Heidelberg, Germany
andreas.dafferner@adw.uni-heidelberg.de

Karin Dornacher, DANTE e.V,
Heidelberg, Germany
office@dante.de

Willi Egger, BOEDE,
Sambeek, The Netherlands
w.egger@boede.nl

Kai Eigner, Tat Zetwerk,
Utrecht, The Netherlands
eigner@tatzetwerk.nl

Ivo Geradts, Tat Zetwerk,
Utrecht, The Netherlands
geradts@tatzetwerk.nl

Frans Goddijn,
Amsterdam, The Netherlands
frans@goddijn.com

Patrick Gundlach, Dante e.V,
Berlin, Germany
patrick@gundla.ch

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

Tomás Hála, Mendel University Brno,
Brno, Czech Republic
thala@mendelu.cz

Taco Hoekwater, Bittext,
Breskens, The Netherlands
taco@bittext.nl

Karel Horak,
Lety, Czech Republic
horakk@math.cas.cz

Jean-Michel Hufflen, University of Franche-Comté,
Besançon Cedex, France
jmhuffle@femto-st.fr

Bogusłav Jackowski, GUST,
Gdañsk, Poland
jacko@bop.com.pl

Hironori KITAGAWA,
Tokyo, Japan
h_kitagawa2001@yahoo.co.jp

Harald König,
Balingen, Germany
koenig@linux.de

Reinhard Kotucha, Capical GmbH,
Hannover, Germany
reinhard.kotucha@web.de

Siep Kroonenberg, RUG,
Groningen, The Netherlands
siepo@cybercomm.nl

Silke Krumrey, Fachhochschule Stralsund,
Stralsund, Germany
silke.krumrey@fh-stralsund.de

Jan Kula,
Prague, Czech Republic
jano.kula@gmail.com

Yusuke KUROKI,
Yokohama, Japan
kuroky@users.sourceforge.jp

Johannes Küster, typoma GmbH,
Holzkirchen , Germany
info@typoma.com

Dag Langmyhr, University of Oslo,
Oslo, Norway
dag@ifi.uio.no

Lucien Lemmens,
Laakdal, Belgium
lcnlmmns@me.com

Manfred Lotz,
Frankfurt, Germany
manfred@dante.de

Jerzy Ludwichowski, GUST,
Toruñ, Poland
jerzy.ludwichowski@umk.pl

Bernd Militzer,
Kempen, Germany
bernd@militzer.net

Christina Möller, Fachhochschule Stralsund,
Stralsund, Germany
christina.moeller@fh-stralsund.de

Thomas Ratajczak, German Army,
Langenfeld, Germany
ratajczak@gmail.com

Heiner Richter, Fachhochschule Stralsund,
Stralsund, Germany
heiner.richter@fh-stralsund.de

Edgaras Šakuras, VTEX UAB,
Vilnius, Lithuania
edgaras.sakuras@vtex.lt

Luigi Scarso,
Padova, Italy
luigi.scarso@gmail.com

Volker Schaa, Dante e.V,
Darmstadt, Germany
v.r.w.schaa@gsi.de

Martin Schröder,
Duisburg, Germany
martin@oneiros.de

Robbert Schwippert, Docwolves B.V.,
Dordrecht, The Netherlands
ras@elvenkind.com

Martin Sievers, Dante e.V.,
Trier, Germany
martin@dante.de

Linas Stonys, VTEX UAB,
Vilnius, Lithuania
lstonys@vtex.lt

Piotr Strzelczyk, GUST,
Gdynia, Poland
piotr@eps.gda.pl

Sigitas Tolušis, VTEX UAB,
Vilnius, Lithuania
sigitas@vtex.lt

Kees Van der Laan,
Garnwerd, The Netherlands
kisa1@xs4all.nl

Ulrik Vieth,
Stuttgart, Germany
ulrik.vieth@arcor.de

Mari Voipio, Lucet.fi,
Vantaa, Finland
mari.voipio@lucet.fi

Herbert Voß, Freie Universität Berlin,
Berlin, Germany
herbert@dante.de

Munehiro YAMAMOTO,
Japan
munepixyz@gmail.com

Uwe Ziegenhagen, DB Private Equity,
Cologne, Germany
ziegenhagen@gmail.com