

Stretching letter and slanted-baseline formatting for Arabic, Hebrew, and Persian with ditroff/ffortid and Dynamic POSTSCRIPT Fonts

DANIEL BERRY (دانيال بيرى, דניאל ברי, دانيال بيرى)

Computer Science Department
Technion
Haifa 32000
Israel

SUMMARY

This paper describes an extension to ditroff/ffortid, a system for formatting bi-directional text in Arabic, Hebrew, and Persian. The previous version of the system is able to format mixed left-to-right and right-to-left text using fonts with separated letters or with connecting letters and only connection stretching, achieved by repeating fixed length baseline fillers. The latest extension adds the abilities to stretch letters themselves, as is common in Arabic, Hebrew, and Persian calligraphic printing, and to slant the baselines of words, as is common in Persian calligraphic printing. The extension consists of modifications in ffortid that allow it to interface with (1) dynamic POSTSCRIPT fonts to which one can pass to the outline procedure for any stretchable and/or connected letter, parameters specifying the amounts of stretch for the letter itself and/or for the connecting parts of the letter and (2) POSTSCRIPT fonts whose characters are slanted so that merely applying show to a word ends up printing that entire word on a single slanted baseline.

As a self-test, this paper was formatted using the described system, and it contains many examples of text written in Arabic, Hebrew, and Persian.

תקציר

מאמר זה מתאריך חבה של ditroff/ffortid, מערכת סדר-דפוס של כתב דו-כיווני בע-בית, ועב-ית-דפ-סית. הגרסה הקודמת של המערכת יכולה לסדר כתב משמאל לימין ומימין לשמאל, בשימוש של גופנים עם אותיות תפ-דות, או עם אותיות מוחברות ומתיחת הקשרים בלבד המתאפשרת באמצעות תחודה על מילוי קו-בסיס בארכים קבועים. וההחבה החדשה מאפשרת את מתיחת האותיות עצמן, כמקובל בדפוס קלי-פי בשפות ע-בית, ועב-ית-דפ-סית, וכן את הטיית קו-בסיס של מילים, כמקובל בדפוס קלוג-פי בשפ-ת-פ-סית. וההחבה מכיל ה שינוי ב-ffortid, המאפשרי מימשק ע-ם (1) גופני דינמיים ב-POSTSCRIPT, אשר ניתן להעב-ר אל שג-רת קו הגבול עב-ר כל אות נמתחת ו/או מקשר-ת-רפ-מ-ש-ים שמגדירים את מידת המתיחה של האות עצמה ו/או של חלקי האות המקשרים (2) גופנים ב-POSTSCRIPT אשר אותיותיהם מוטות, כד-ש-ק פנייה אל show חצית הדפסה של המילה השלמה על קו-בסיס מוטה. הל-צ-ו-ך בדיקה עצמית, מאמר זה נסד-ר באמצעות המערכת שתוארה לעיל, והוא כולל דוגמאות-בות של כתב בע-בית, ועב-ית-דפ-סית.

خلاصة

هذه المقالة تصف توسيع ditroff/ffortid، برنامج تصفيف ذو-إتجاهين للغات العربية العبرية، والفارسية. النسخة السابقة

للبرنامج قدرة على تصنيف نص مختلط من اليسار-إلى-اليمين ومن اليمين-إلى-اليسار. باستعمال أحرف منفصلة أو أحرف متصلة وفقط مد الاتصال، بواسطة إعادة طباعة وصلات ذات طول نانت. التوسيع الأخير يضيف المقدرة على مدّ الأحرف نفسها، كما هو مُتَّبَعُ في فنون الخطوط الاعربية، العبرية، والفارسية، وبالإضافة إلى إمالة قاعدة الكلمات، كما هو مُتَّبَعُ في الخط الفارسي. الأضافة مبنية على تغييرات ل-ffortid والتي تسمح لها بالاتصال مع (١) أحرف PostScript متغيرة (dynamic) والتي بالأمكن تمديدها لمعادلة القاعدتولي حرف بالمكان مدّه أو/و وصله، مع متغيرات تصف كمية مد الحرف نفسه أو/و الاحزاي التي تصل الأحرف ببعضها، و (٢) أحرف PostScript مائلة لدرجة show أن طباعة الكلمة تظهر على قاعدة مائلة واحدة.

كإختبار ذاتي للبرنامج، هذه المقالة طُبِعَت بواسطة البرنامج الموصوف أعلاه، وتحتوي على عدة أمثال لنُصص كُتبت باللغات العربية، العبرية، والفارسية.

خلاصه

این مقاله بسطی بر ditroff/ffortid را تشریح میکند که سیستمی است برای نقشبندی دو جهتی نوشته‌ها به عربی، عبری، و فارسی. ورژن پیشین این سیستم میتواند نوشته‌های مخلوط از چپ به راست و از راست به چپ را با استفاده از حروف جدا از هم یا بهم پیوسته، و تنها بوسیله کشیدن اتصالها با تکرار پرکن‌های پایه‌خطی با طول ثابت، نقشبندی کند. آخرین بسط این سیستم قابلیت‌های کشیدن خود حروف، که در چاپ خطاطی عربی، عبری، و فارسی معمول است، و کژ کردن پایه‌خط کلمات، که در چاپ خطاطی فارسی معمول است، را به آن اضافه میکند. این بسط متشکل از تغییراتی است در ffortid که به آن دو امکان زیر را میافزاید: (١) استفاده از خط‌های دینامیک

POSTSCRIPT که میتوان به آنها برنامه پوش (قالب) هر حرف کشیدنی یا اتصالی را داد، بعلاوه پارامترهایی که اندازه کشیدن هر حرف یا هر بخش اتصالی هر حرف را مشخص میکنند؛ و (۲) استفاده از خطهای خمیده POSTSCRIPT بطوریکه انجام دستور show برلی یک کلمه به تنهایی باعث چاپ تمام آن کلمه بر روی یک پایه خط خمیده یگانه میشود.

بعنوان یک خودآزمایی، این مقاله با استفاده از سیستمی که در اینجا تشریح میشود نقشبندی شده و مثالهایی از نوشته‌های عربی، عبری، و فارسی در متن خود دارد.

خلاصه

این مقاله بسطی بر ditroff/ffortid را تشریح میکند که سیستمی است برای نقشبندی دو جهتی نوشته‌ها به عربی، محاسب، و فارسی. ورژن پیشین این سیستم میتواند نوشته‌ها مخلوط از چپ به راست و از راست به چپ را با استفاده از حروف جدا از هم یا بهم پیوسته، و تنها بوسیله کشیدن اتصالات با تکرار پرکرها پایه خطی با طول ثابت، نقشبند کند. آخرین بسط این سیستم قابلیتهای کشیدن خود حروف، که در چاپ خطاطی عربی، محاسب، و فارسی معمول است، و کژ کردن پایه خط کلمات، که در چاپ خطاطی فارسی معمول است، را به آن اضافه میکند. این بسط متشکل از تغییراتی است در ffortid که به آن دو امکان زیر را میافزاید: (۱) استفاده از خطها دینامیک POSTSCRIPT که میتوان به آنها برنامه پوش (قالب) هر حرف کشیدنی یا اتصالی را داد، بعلاوه پارامترهایی که اندازه کشیدن هر حرف یا هر بخش اتصالی هر حرف را مشخص میکنند؛ و (۲) استفاده از خطها خمیده POSTSCRIPT بطوریکه انجام دستور show برلی یک کلمه به تنهایی باعث چاپ تمام آن کلمه بر روی یک پایه خط خمیده یگانه میشود.

بعنوان يك خود آزمائي، اين مقاله با استفاده از سيستمي كه در اينجا تشریح ميشود نقشه‌بند شده و مثالهایی از نوشته‌ها عربي، محاسب، و فارسي در متن خود دارد.

KEY WORDS Arabic Bidirectional Formatting Multilingual Troff Stretching Keshide Slanted-baseline

1 INTRODUCTION

The reader is assumed to be familiar with the paper “Arabic Formatting with ditroff/ffortid” Published in *Electronic Publishing* in December, 1992 [1]. This paper described an Arabic formatting system that is able to format multilingual scientific documents that contains text in Arabic or Persian, as well as other languages, plus pictures, graphs, formulae, tables, bibliographical citations, and bibliographies. The system is an extension of ditroff/ffortid that is already capable of handling Hebrew in the context of multilingual scientific documents. ditroff/ffortid itself is a collection of pre- and postprocessors for the UNIX ditroff (Device Independent Typesetter RunOFF) [2] formatter. The system was built without changing ditroff itself. The extension consists of a new preprocessor, fonts, and a modified existing postprocessor.

The preprocessor transliterates from a phonetic rendition of Arabic using only the two cases of the Latin alphabet. The preprocessor assigns a position, stand-alone, connected-previous, connected-after, or connected-both, to each letter. It recognizes ligatures and assigns vertical positions to the optional diacritical marks. The preprocessor also permits input from a standard Arabic keyboard using the standard ASMO [3] encoding. In any case, the output has each positioned letter or ligature and each diacritical mark encoded according to the font’s encoding scheme.

The fonts are assumed to be designed to connect letters that should be connected when they are printed adjacent to each other.

The postprocessor is an enhancement of the ffortid [4] program that arranges for right-to-left printing of identified right-to-left fonts. The major enhancement is stretching the final connection to letters of lines or words, using a fixed-sized baseline filler, instead of inserting extra inter-word spaces, in order to justify the text.

This ffortid does not handle letter stretching, and the Srouji and Berry paper observes in its conclusion that adding the ability to stretch letters would be left for future work. That future work, applied to Arabic, Hebrew, and Persian, together with the abilities to stretch connections without a filler and to slant the baseline of words as in Persian is the subject of the present paper.

As usual for papers dealing with formatting issues, this paper was typeset with the software described herein. The commands that cause the printing of this paper are:

```
refer -d -e -n -p /home/dberry/.refsidx paper > paper.ref
```

```
psfig paper.ref|sed -f cross.references|tbl|eqn -Tpsc | \
troffort -mXP -r13 -r14 -r22 -r42 -r43 -r44 -r45 \
--13 --42 --43 --44 --45 -sea -msw \
-IhD -IAN -IPN -IPU -pn -IhF -ILG -t > $F.ps
```

where mounted on position

- 13 is a stretchable Hebrew font called Frank Ruehl (hF),
- 14 is a non-stretched mounting of that on position 13,
- 23 is a non-stretchable Hebrew font called David (hD), which is also not reversed,
- 42 is a stretchable Arabic font called Naskh (AN),
- 22 is a non-stretched mounting of that on position 42,
- 21 is a non-stretched mounting of that on position 42, which is also not reversed,
- 43 is a stretchable slanted Persian font called Persian-Naskh (PN) with no space between words,
- 44 is a stretchable non-slanted Persian font called Persian-Unslanted (PU) with space between words,
- 45 is an unslanted version of that on position 43 (pn), and
- 20 is the Logo font (LG) for the word METAFONT.

2 STRETCHING

In high quality Arabic and Persian printing, stretching is preferred to inserting extra inter-word spacing to achieve left (end-of-the-line) justification. Figure 1 shows an example from an Egyptian textbook in Arabic calligraphy [5, 6]. This stretching is called “*keshide*” in both Arabic and Persian from the Persian word *keshidan*, which means “to stretch”. In Arabic and Persian, there are two kinds of stretching,

1. stretching of the connection to the last connecting-before letter in a line or words, and
2. stretching of the last stretchable letter in a line or words.

In Figure 1, the pronounced stretching at the end of line 2 is of a letter, the connecting-before *nun*, while those near the middles of lines 3 and 4 are of connections.

In high quality Hebrew printing, especially in the Torah and in prayer books, stretching is for the same purpose, to achieve justification without spreading words. Figure 2 shows some stretching in the first verses from the book of Genesis [7]. Since Hebrew letters are not connected to each other, only letter stretching is available and it is usually applied to the last stretchable letter in a line. One sees in line 4 a stretched \beth (*bet*), in line 20 a stretched η (*he*), in lines 5, 10, 15, 16, 18, and 19 stretched resh (*reshes*), all of differing amounts, and in lines 9 and 13 stretched \daleth (*lameds*), of differing amounts. Because the amounts of stretch are all different, a collection of stretched versions of letters of specific amounts of stretch is not likely to work. (All occurrences of words for G-d or G-d’s name have been removed in order to avoid profaning these words).

One can imagine a scribe in these languages, writing with indelible ink on expensive parchment, not having an opportunity to rewrite a line to left justify it after it is known what text can fit in the line. Stretching is applied near the end of the line, when it is easier to predict how much more can be written on the line. The scribe merely stretches

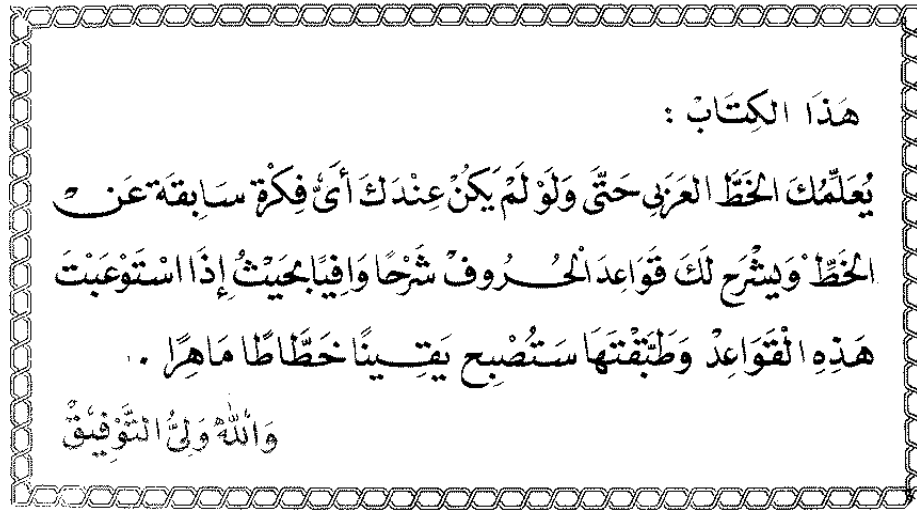


Figure 1: Sample of Arabic Calligraphic Writing.

either the last connection or the last stretchable letter as is available. It is the perfect way to achieve justification when there is only one pass over the material and there is no backtracking. Of course, with modern formatting software, it is not difficult to achieve justification by spreading words. However, old habits die hard and still stretching is associated with high quality printing. Furthermore, not all stretching is applied at the end of the line in a one-pass mode. There are numerous examples of printing in which the scribe was careful to balance the stretching over several points in the line to achieve a pleasing appearance, as is shown in Figure 3, taken from the same text book from which Figure 1 was taken. It shows the letters ب, ف and a variation of ك, without and with stretching.

The unstretched versions of the letters are said to be 5 points wide (the point is the width of the dot that appears in two of the letters), and the stretched versions are 11 points wide. This figure also shows the importance of aesthetics in stretching, an importance that precludes simple algorithms. Because the three letters that are stretched are structurally similar, for appearance's sake, they had to be stretched the same amount rather than individual amounts according to the needed justification. For this kind of situation, the human calligrapher must exercise lookahead.

3 STRETCHING PROBLEMS AND DYNAMIC FONTS

The following discussion is couched in terms of POSTSCRIPT [8] fonts, although the problems exist and the proposed solutions are available also in other outline-based font formats such as TrueType [9].

A form of connection stretching is quite easy to achieve with most available fonts. It requires only that

וְהָאָרֶץ הִיְתָה תְהוֹ וּבְהוּ וְחֻשֶׁךְ עַל פְּנֵי הַתְהוֹ וּוְיֹזֶז
 מִרְחֹפֶת עַל פְּנֵי הַמַּיִם וַיֹּאמֶר אֱלֹהִים יְהִי
 אֹר וְיִהְיֶה אֹר וַיֵּרָא אֶת הָאֹר כִּי טוֹב
 וַיִּבְרָא בֵּין הָאֹר וּבֵין הַחֹשֶׁךְ וַיִּקְרָא
 לְאֹר יוֹם וְלַחֹשֶׁךְ לַיְלָה וַיְהִי עֶרֶב
 וַיְהִי בֹקֶר יוֹם אֶחָד
 וַיֹּאמֶר יְהִי רִקְעַת בְּתוֹר הַמַּיִם וְיִהְיֶה מַבְרֵיחַ
 בֵּין מַיִם לַמַּיִם וַיַּעַשׂ אֶת הַרְקִיעַ וַיִּבְרָא
 בֵּין הַמַּיִם אֲשֶׁר מִתַּחַת לַרְקִיעַ וּבֵין הַמַּיִם אֲשֶׁר
 מִעַל לַרְקִיעַ וַיְהִי כֵן וַיִּקְרָא לַרְקִיעַ שָׁמַיִם
 וַיְהִי עֶרֶב וַיְהִי בֹקֶר יוֹם שֵׁנִי
 וַיֹּאמֶר יִקְווּ הַמַּיִם מִתַּחַת הַשָּׁמַיִם אֶל
 מְקוֹם אֶחָד וְתִרְאֶה הַיַּבֵּשֶׁת וַיְהִי כֵן וַיִּקְרָא
 לַיַּבֵּשֶׁת אֶרֶץ וּלַמְּקוֹה הַמַּיִם קִיָּה יַמִּים וַיֵּרָא
 כִּי טוֹב וַיֹּאמֶר תִּדְשֵׂא הָאָרֶץ
 דֶּשֶׁא עֵשֶׂב מִזְרִיעַ זֶרַע עֵץ פְּרִי עֵץ פְּרִי לְמִינֵהוּ
 אֲשֶׁר זֶרְעוֹ בּוֹ עַל הָאָרֶץ וַיְהִי כֵן וַתִּבְרָא הָאָרֶץ
 דֶּשֶׁא עֵשֶׂב מִזְרִיעַ זֶרַע לְמִינֵהוּ וְעֵץ עֵץ פְּרִי
 אֲשֶׁר זֶרְעוֹ בּוֹ לְמִינֵהוּ וַיֵּרָא כִּי טוֹב וַיְהִי
 עֶרֶב וַיְהִי בֹקֶר יוֹם שְׁלִישִׁי

Figure 2: Sample of Hebrew Calligraphic Writing.

1. all connecting letters connect via baseline strokes which meet their bounding boxes at the same y coordinates
2. that the show command place the characters to be printed bounding box to bounding box, and
3. that the font provide a narrow rectangular baseline filler that connects on both sides to the connecting baseline of all connecting letters.

Figure 4 shows the connecting after and the connecting before forms of *baa* connected to each other, then disconnected, then with two fillers in between them but disconnected, finally with the two fillers and all connected.

Letter stretching is much more difficult to achieve, and that difficulty is probably the reason that it does not seem to appear in any of the available Arabic formatting systems.

One possible way to achieve letter stretching is for the font to provide additional

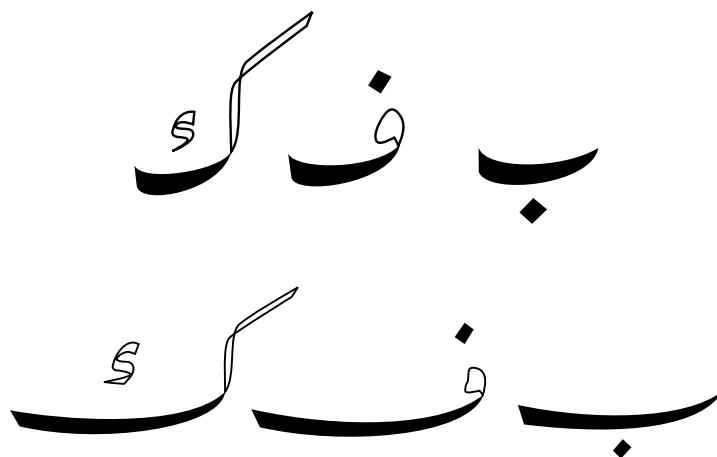


Figure 3. Non-stretched and stretched letters.

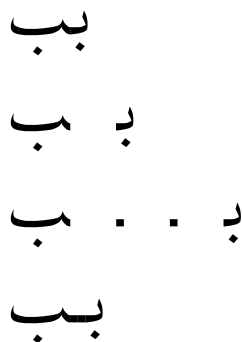


Figure 4: Baas with stretched connections.

stretched forms of each stretchable letter, each form being the stretching of one letter by a specific amount. The problem with this approach is the large number of additional letters required in the font to provide enough different sizes of each stretchable letter to cover most needs. Arabic fonts are usually quite full, taking up nearly all 256 available positions with all the letters and ligatures and the up-to-four forms of each letter and ligature. Moreover, there will always be specific stretching needs that are not covered by the available collection.

An alternative, and probably better approach would be to make the procedure for each stretchable letter accept a parameter whose value, defaulted to zero, indicates the amount of stretch with which the outline of the letter is to be drawn. When it is necessary to stretch a letter by any specific amount, the `show` of that letter passes the amount to the letter's procedure and informs the font mechanism of the new length of the letter.

The problem with the second approach is that it requires dynamic fonts, as proposed

by J. André and B. Borghi [10]. Such dynamic fonts violate the basic assumption for type 1 fonts [11], namely that for any given character code, scale (point size), and font name, the bitmap obtained by executing the character's procedure is constant. The holding of this assumption allows caching the bitmap so that it does not need to be computed again when a request to print the same character in the same scale and font comes later. Dynamic fonts thus print slower because it is useless to cache a character's bitmap when each time it is printed, it will have a different amount of stretch and thus a different bitmap. Such dynamic fonts need to be type 3, that is, allowing use of the full POSTSCRIPT language, and thus are slower to print and lose other benefits of type 1, i.e., the ability to provide hints to improve low resolution or small point size rasterization and the ability to be handled by the Adobe Type Manager (ATM). However, in this author's opinion, the beauty of what can be done with dynamic fonts, including dynamic optical scaling [12], outweigh the disadvantages. Moreover, as the use of dynamic fonts grows, Adobe will be provided an incentive to extend hinting and the ATM mechanism to cover the needed dynamism, perhaps through a type 2 font providing just enough capability from the full POSTSCRIPT to cover most needs for dynamic fonts. Note that multi-master fonts (MMFs) do not fill the bill. Each can be thought of as a font definition macro, which when supplied with widths of the font to be simulated, generates a proper type 1 font definition, each of whose characters prints to the specified width.

4 OPERATION OF `ffortid`

As shown in Figure 5, `ffortid` sits between `ditroff` and the device driver and converts from the `ditroff` intermediate form to the same form so that the device driver does not know that it did not get its input directly from `ditroff`.

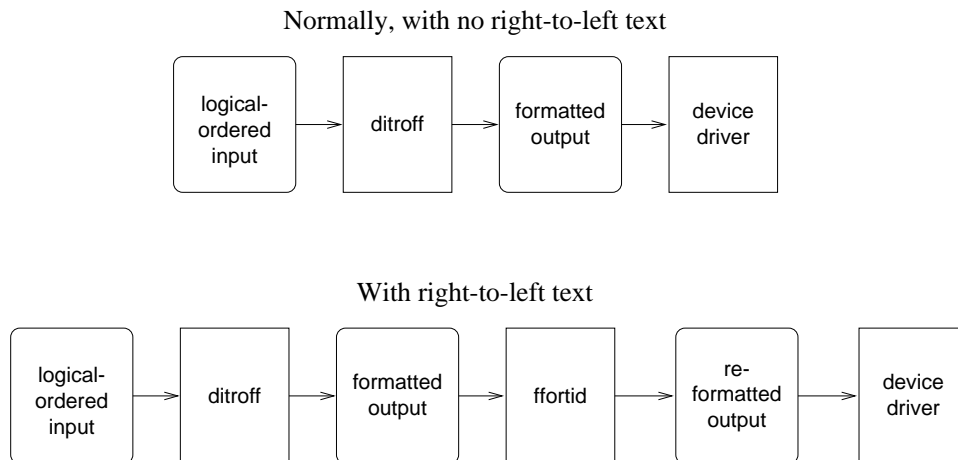


Figure 5: Flow of `ffortid` and related programs.

Recall that the `ditroff` intermediate form, besides showing the exact position of each character to be printed, has a specific marker at the end of each output line and a specific marker at the end of each input word. A program processing this form does not have to

guess, and possibly be wrong, where the ends of lines and ends of words are. Recall finally, that the characters come out of *ditroff* broken into lines but in logical order [13, 14, 4], that is from left to right in the order that one hears the characters as they are spoken.

The main job of *ffortid* is to reorganize the characters so that the text is in visual order, in which the text of each language is written in its own direction and the flow of the single-directional chunks in each line is consistent with the *current document direction*. *ffortid* works by totally reformatting each line as a function of the current document direction and the font of each character. At the top level of abstraction, it reads the characters of each line, delimited by the end-of-line marker, and permutes characters so that the single-directional chunks of a line flow in the current document direction which the characters in each chunk flow in the chunks single direction.

```

for each line in the file do
  if the current document direction is LR then
    reverse each contiguous sequence of RL characters in the line
  else (the current document direction is RL)
    reverse the whole line;
    reverse each contiguous sequence of LR characters in the line
  fi
od

```

An *RL (LR) character* is a character in any RL (LR) font. This simple procedure falls flat on its face when presented with an embedded LR numeral inside RL text inside a LR document, e.g., inside an English document, a Hebrew address containing a Latin house number; the LR numeral splits the RL text into two pieces and the two pieces end up being in the document's LR order relative to each other rather than the required RL order relative to each other. This anomaly is prevented by applying the algorithm recursively on the RL text. Before the introduction of the stretching capability, the original *ffortid*, as a last step, applied *ditroff*'s normal word-spreading algorithm to increase the inter-word white space in each line to cause the text to be justified at both margins. To be concrete, without this provision, the time-ordered input:

He lives at חיפה 15בא49 אישנה in Israel.

would appear as:

He lives at חיפה 15בא49 הנשיא in Israel.

instead of the correct:

He lives at חיפה 15בא49 הנשיא in Israel.

Note that the logical ordering of the house number is "4-9-alef-bet-1-5". and that the this number must come *after* the name of the street, הנשיא and *before* the name of the city חיפה in the right-to-left flow of the Hebrew text that is embedded in an English sentence in a left-to-right document.

It is clear that the total amount of white space between the words in a line, δ , is calculable as the difference between the current line length and the sums of the lengths of the words and the minimal spacing. Stretching is achieved by not spreading the words and dividing δ by the number of stretching places to get an amount, σ , to stretch at each stretching place. When connection stretching is used, then a number of these fillers whose total length is greater than equal to σ is put in each connection to be stretched. The last of these can be made to overlap its predecessor so that its end is precisely at the beginning of the character to which it connects. Clearly, if stretching of letters is added, then the simple strategy is to pass this σ amount as the length increase parameter to the letters to be stretched. Therefore the problem is to modify the POSTSCRIPT definitions of the fonts so that the procedure for a stretchable letter takes the amount of stretch as a parameter and modifies the outline appropriately.

5 DYNAMIC FONTS

With the general strategy laid out, the step of highest technological risk was the production of the dynamic fonts with the parameterized stretchable characters. Hence, this was the first step to work on.

In Arabic, Hebrew, and Persian, all stretching is across horizontal portions of a letter such that stretching them does not cause the result to look like another letter. Letters, such as the alif in Arabic and Persian and the vav in Hebrew, that are entirely vertical cannot be stretched. In Hebrew, all horizontal portions involve sections bounded by parallel horizontal straight line segments. In Arabic and Persian, the horizontal portions are not straight; they involve smooth curves that are mostly horizontal. Thus stretching Hebrew letters is rather straightforward, but stretching Arabic and Persian letters involves stretching curves in such a way that the curvature, in the vernacular sense of the word, is somehow preserved and that the stretched sections flow smoothly to the rest of the letter. Therefore, it was decided to build a stretchable Hebrew font first. This way, the details of the parameterization could be worked out independently of dealing with the aesthetics of curve stretching.

6 DYNAMIC HEBREW FONTS

The stretchable Hebrew letters are those with horizontal portions for which stretching the horizontal portion would not cause the result to look like another letter. Therefore, the stretchable letters are ב (*bet*), ד (*daled*), ה (*heh*), כ (*khet*), ך (*khaph softit*), ק (*kaph*), ל (*lamed*), ם (*mem softit*), ן (*nesh*), and ת (*tav*). This author has seen א (*aleph*), מ (*mem*), and ף (*quf*) stretched in some documents. Note that ו (*vav*) is not stretchable because stretching its small horizontal portion would cause the result to look like ן (*nesh*).

It was decided to stick to the definite set for now. Once the technique is understood, it is easy to add additional stretchable letters. Indeed, the software has been carefully designed so that the parts affected by adding additional letters to the stretchable set are completely table-driven from human-editable ASCII tables.

First it was necessary to decide how to pass the amount of stretch to each letter to be stretched. It was decided that before the `show` of the letter to be stretched, a global variable called `fact` would be set to the amount of stretch. The procedure for this letter could use this value as it sees fit to adjust the values of mostly x coordinates in the outline

path. Following this show, `fact` would be reset to zero, with the intention that a stretchable letter finding `fact` set to zero would end up not being stretched. Initially, the units of `fact` were those of the `FontMatrix` of the font. However, such a unit requires the invoker to know the `FontMatrix` of the font, which is normally considered a hidden implementation detail. Later, it was decided to make the unit of `fact` be emms. The letter's procedure is required to convert the value of `fact` into `FontMatrix` units by multiplying `fact` by the x value in the `FontMatrix`, and does so with the POSTSCRIPT code

```
/fact fact Name-of-font-definition-dictionary
  /FontMatrix get 0 get div def
```

To stretch a character, it is necessary to examine its outline and to find the points defining the straight line segments to be stretched. The simplest situation is when the definition of a character's outline does an absolute move to a starting point and then draws the path using only relative commands thereafter. Recall that all stretching in Hebrew is across straight line segments; if the path is described with relative commands, each such segment is defined with an `rlineto` whose prefix arguments are the x and y of the ending point, the starting point assumed to be where the current point is at the start of the command. Therefore, it suffices to decide which `rlineto` segments are to be stretched and to add or subtract `fact`, depending on the direction in which the segment is being drawn, to the x argument of the `rlineto`. Thus, the definition of the Hebrew letter *bet* from one font,

```
/bet {
464 0 -61 -26 482 541 setcachedevice
0 0 moveto
127 388 rmoveto
-127 0 rlineto
-2 0 35 52 44 62 rcurveto
9 11 54 67 54 65 rcurveto
0 -47 rlineto
22 0 rlineto
114 0 rlineto
30 0 80 0 107 -32 rcurveto
25 -30 19 -90 19 -104 rcurveto
0 -252 rlineto
61 0 rlineto
-61 -80 rlineto
-360 0 rlineto
56 80 rlineto
233 0 rlineto
0 0 -1 221 0 238 rcurveto
0 9 0 42 -9 56 rcurveto
-9 15 -28 14 -46 14 rcurveto
-107 0 rlineto
fill } def
```

is converted to

```

/bet {
%%464 0 -61 -26 482 541 setcachedevice
543 fact add 0 setcharwidth
0 0 moveto
127 fact add 388 rmoveto
-127 fact sub 0 rlineto
-2 0 35 52 44 62 rcurveto
9 11 54 67 54 65 rcurveto
0 -47 rlineto
22 0 rlineto
114 fact add 0 rlineto
30 0 80 0 107 -32 rcurveto
25 -30 19 -90 19 -104 rcurveto
0 -252 rlineto
61 0 rlineto
-61 -80 rlineto
-360 fact sub 0 rlineto
56 80 rlineto
233 fact add 0 rlineto
0 0 -1 221 0 238 rcurveto
0 9 0 42 -9 56 rcurveto
-9 15 -28 14 -46 14 rcurveto
-107 fact sub 0 rlineto
fill } def

```

Observe that the `setcachedevice` command was replaced by a `setcharwidth` command that implements the same net character width. Since a dynamic character may produce different outline each time it is called, its bitmap cannot be cached; if it were, then the next time the character is shown, the procedure would be totally ignored and the previously produced bitmap would be used, possibly of the wrong width. The defined character at point size 30 and a 1 emm stretching of it are

ב ב

The stretching occurs at the boundary between the curves leading to the straight line segments and the straight line segments both upstairs and downstairs. In the case that the path definition is not a series of relative commands, but a series of absolute commands, it is necessary to calculate the x coordinate of all points in terms of the `fact` parameter. Generally, the x s to the right of the stretching are all increased by the value of `fact` and those to the left of the stretching are left alone;

Erez Manor, a student, implemented much of the stretching of the Hebrew font used in this paper as a project for the author's Electronic Publishing seminar.

7 DYNAMIC ARABIC FONTS

There are two parts of Arabic and Persian letters to be stretched, the letter itself in the cases of stand-alone as well as connecting letters, and the connecting parts in the cases of connecting letters only.

7.1 Stretching Letters

In Arabic and Persian, the situation is more complex than in Hebrew. No letter has portions bounded by horizontal straight line segments. There are letters with curved strokes in which the predominant flow of the stroke is horizontal. It is more correct to say that there are forms of letters with these predominantly horizontal curved strokes; not all forms of the same letter have these strokes. Therefore a form is stretchable if it has a predominantly horizontal curved stroke such that stretching it does not yield a result that looks like a form of another letter. On this basis, the stretchable forms are ب (stand-alone *baa*), س (stand-alone *seen*), ص (stand-alone *sad*), ف (stand-alone *faa*), ق (stand-alone *qaf*), ك (stand-alone *caf*), گ (stand-alone *Gaf*), ل (stand-alone *lam*), ن (stand-alone *noon*), ی (stand-alone *alefmaksura*), ک (connecting-after *caf*), گ (connecting-after *Gaf*), ک (connecting-both *caf*), گ (connecting-both *Gaf*), ب (connecting-before *baa*), س (connecting-before *seen*), ص (connecting-before *sad*), ف (connecting-before *faa*), ق (connecting-before *qaf*), ك (connecting-before *caf*), گ (connecting-before *Gaf*), ل (connecting-before *lam*), ن (connecting-before *noon*), ی (connecting-before *alefmaksura*), فی (stand-alone *faa_yaa*), لی (stand-alone *lam_alefmaksura*), لی (connecting-before *lam_alefmaksura*), and all other forms that vary from these by additional or fewer dots or *hamzas*. In particular, ا (stand-alone *alef*) is not stretchable because it has no horizontal strokes.

The outline of any character is a series of elements, each of which is a line, four-point Bézier curve, circle, or arc. As is shown in Figure 6, elements that share an end point p are tangent to the same line which passes through p if the outline is supposed to be smooth through p , and elements that share an end point p have tangents at an angle at p if the outline is supposed to have a corner at p . As a rule, stretching should not introduce, remove, or change any corner. Therefore, tangents at end points of all elements should not change. While horizontal stretching will definitely change x values in the outline, for other reasons, it is not good that y values change. For example, it should be possible to vertically position a diacritical over or under a stretched character as it is positioned over or under the unstretched version. Furthermore, it should not be necessary to have to adjust vertical spacing dynamically as a part of stretching.

With Arabic and Persian, all stretching is across curved portions of letters. Therefore, the issue is how to stretch curves that are described in the font definition as four-point Bézier curves. A four-point Bézier curve is easy to stretch, if as is shown in Figure 7, the stretching is in the interior of the curve between the two middle, control points. To stretch such a curve by A units, simply add A to the x values of the two right hand points

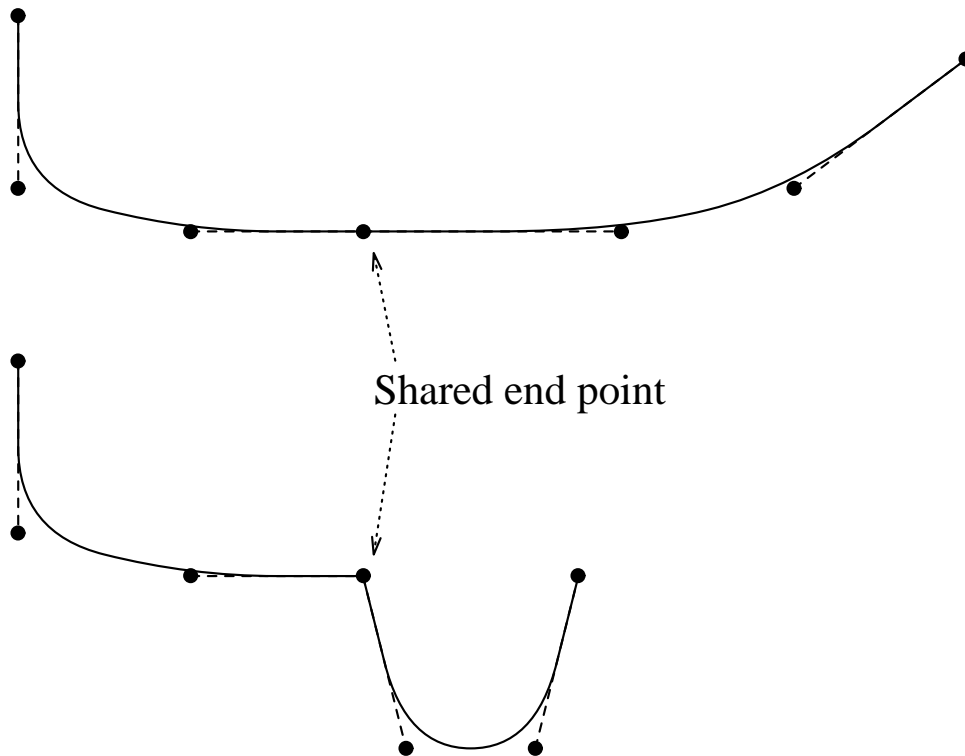


Figure 6: Smooth meeting and cornered meeting.

and to all points to the right of the left hand one of these. Note that the stretched curve connect with its neighbors with the same slope and at the same y values as before. Moreover, it appears that stretching 4-point Bézier curves in the middle in this manner always gives aesthetically pleasing results.

It is a problem to stretch through the shared end point of two Bézier curves whose tangents are the same at the shared point. As is shown in Figure 8, adding A to the x values to the three rightmost points of the right hand curve in order to stretch between the first and second point introduces a corner into what was a smooth meeting at the shared point.

As shown in Figure 9, the corner can be avoided by preserving the slope in the left hand tangent of the right hand curve by increasing both the x and y values by amounts consistent with the slope of the tangent. However, now the right end of the combined curve does not have the same y value as before. One proper solution requires redesign of the two adjacent Bézier curves into one or three adjacent curves so that the place of stretch is in the middle of one 4-point Bézier curve. One special case of stretching through a shared end point works, specifically, when the tangents through the shared end points are completely horizontal, as suggested by Figure 10.

Another solution was discovered during the first experiments, with the connecting-previous form of the *baa*. This turned out to be a case of wanting to stretch across a

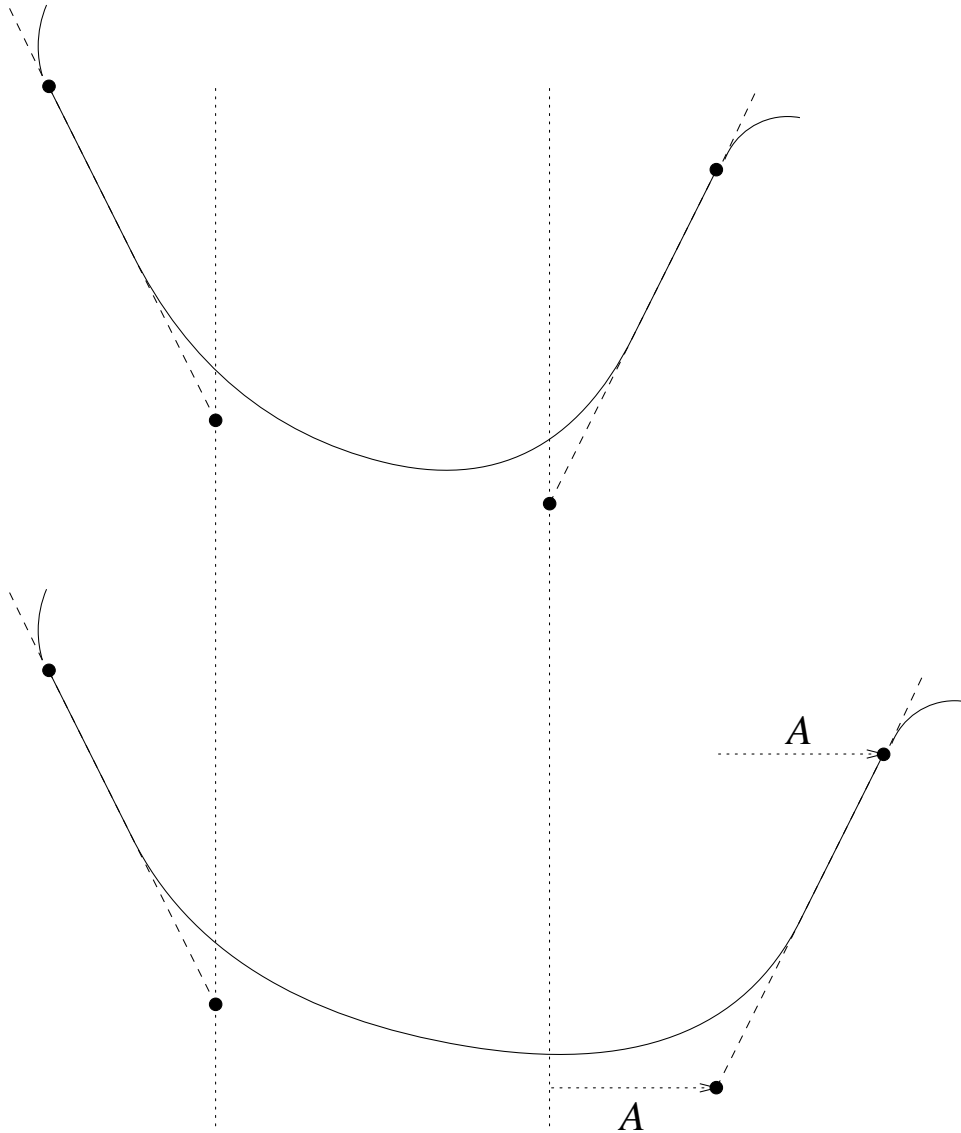


Figure 7: Stretching 4-point Bézier curve.

shared end point when the tangents going into this point are not completely horizontal.

First it was necessary to plot the outline of this letter in order to see the locations of the end and control points of the Bézier curves defining the outline. Figure 11 shows the result of systematically converting the POSTSCRIPT code for the outline of the form into a drawing of its outline with points marked by circles; the starting point of the path that is stretched is labelled with “Start”, the direction of flow is indicated by the arrow coming out of the label, end points of curves and lines are big circles, control points of curves are

little circles, and the stretched curves are marked with vertical arrows.

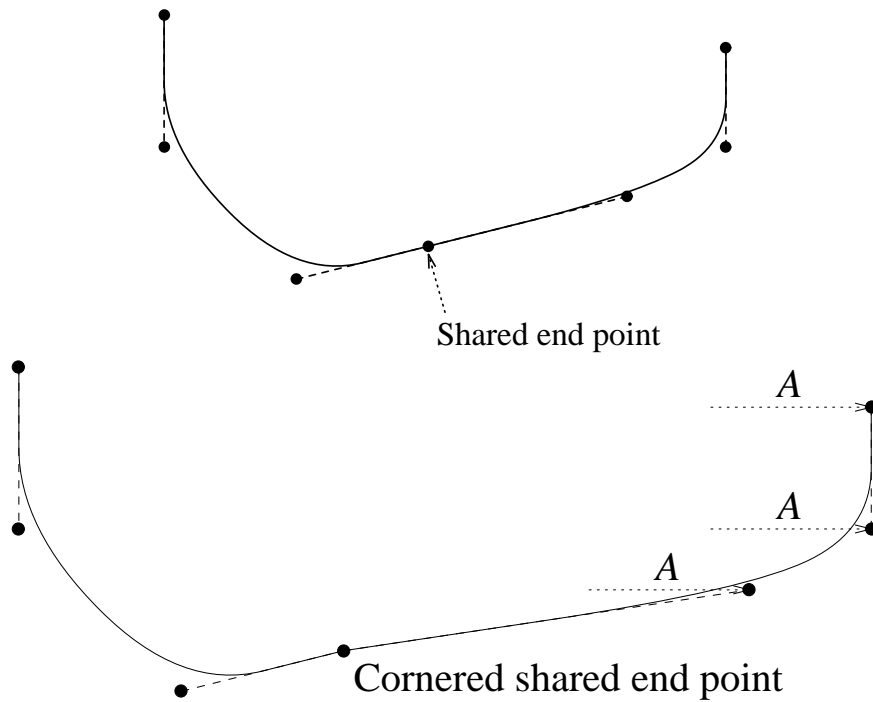


Figure 8: Cornered shared end point.

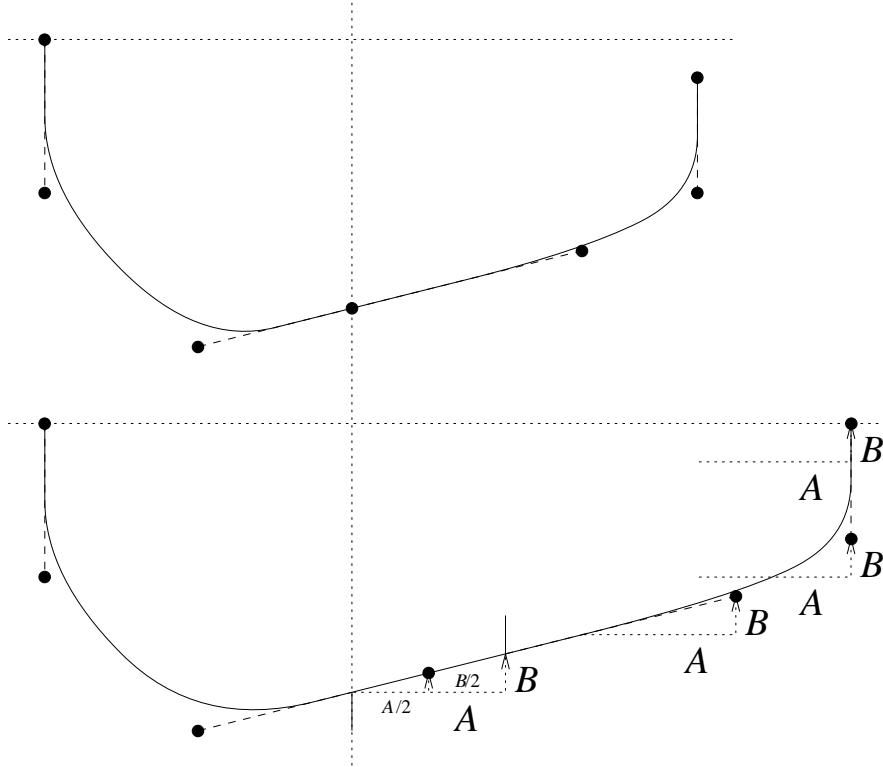


Figure 9: Avoiding cornered shared end point.

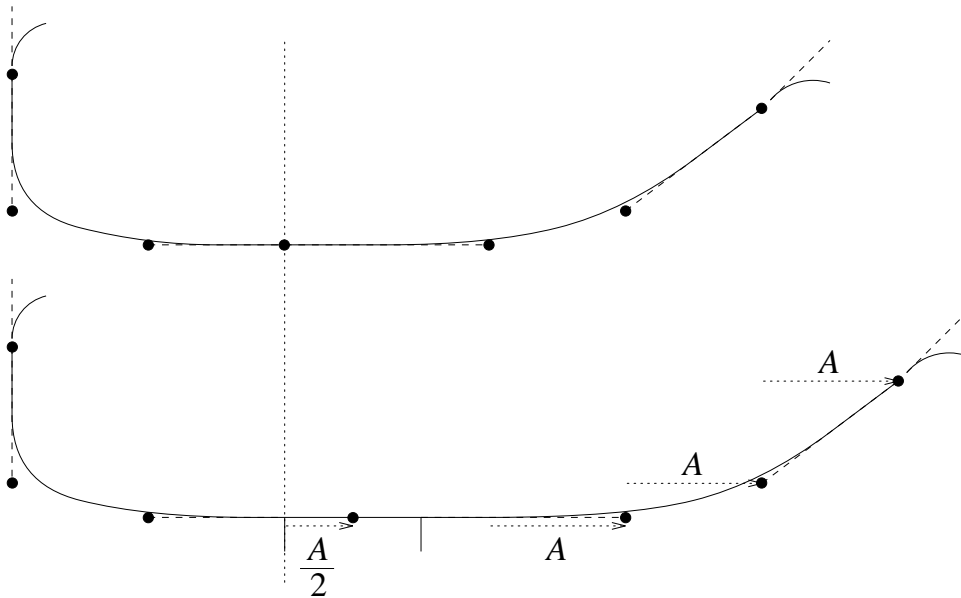


Figure 10: Stretching through shared end point with horizontal tangents.

Each of the upper and lower strokes defining the horizontal portion to be stretched have two Bézier curves meet at a shared end point which is smack dab in the middle of the stroke. The shared end points are marked with asterisks in the figure. Stretching through the upper and lower shared points would introduce corners. Rather than rebuilding the adjacent curves as one curve or adding an additional curve, it was decided to split the amount of stretch A into two equal parts $A/2$, and to stretch each of the adjacent curves in its own middle by $A/2$. The stretched Bézier curves are all marked with pluses in the figure. The result was that the normal code for the outline of the connecting-previous *baa*:

```

/baa_CP
{ 502 0 -15 -180 527 153 setcachedevice
0 0 moveto
243 -94 rmoveto
42 -32 rlineto
-34 -38 rlineto
-40 28 rlineto
closepath
0 0 moveto
260 -5 rmoveto
-17 -1 -136 -10 -192 15 rcurveto
-58 25 -18 106 -8 127 rcurveto
0 -6 -11 -57 11 -69 rcurveto

```

```

67 -39 163 -24 190 -23 rcurveto
28 2 129 10 156 29 rcurveto
17 11 27 44 28 48 rcurveto
1 -14 3 -27 28 -51 rcurveto
20 -19 27 -15 29 -15 rcurveto
0 -56 rlineto
-1 0 -21 -6 -48 26 rcurveto
-9 11 -16 17 -22 34 rcurveto
0 -1 1 -16 -28 -34 rcurveto
-46 -28 -86 -25 -144 -31 rcurveto
closepath 0 0 moveto fill } def

```

is converted into the following code that uses `fact` initialized to the amount of stretch in emms (with the changes emboldened).

```

/baa_CP
{ /fact fact Arabic-NaskhFont /FontMatrix get 0 get div def
502 fact add 0 setcharwidth
/fact_2 fact 2 div def
0 0 moveto
243 fact_2 add -94 rmoveto
42 -32 rlineto
-34 -38 rlineto
-40 28 rlineto
closepath
0 0 moveto
260 fact_2 add -5 rmoveto
-17 -1 -136 fact_2 sub -10 -192 fact_2 sub 15 rcurveto
-58 25 -18 106 -8 127 rcurveto
0 -6 -11 -57 11 -69 rcurveto
67 -39 163 fact_2 add -24 190 fact_2 add -23 rcurveto
28 2 129 fact_2 add 10 156 fact_2 add 29 rcurveto
17 11 27 44 28 48 rcurveto
1 -14 3 -27 28 -51 rcurveto
20 -19 27 -15 29 -15 rcurveto
0 -56 rlineto
-1 0 -21 -6 -48 26 rcurveto
-9 11 -16 17 -22 34 rcurveto
0 -1 1 -16 -28 -34 rcurveto
-46 -28 -86 fact_2 sub -25 -144 fact_2 sub -31 rcurveto
closepath 0 0 moveto fill } def

```

The defined character at point size 30 and a 1 emm stretching of it are:



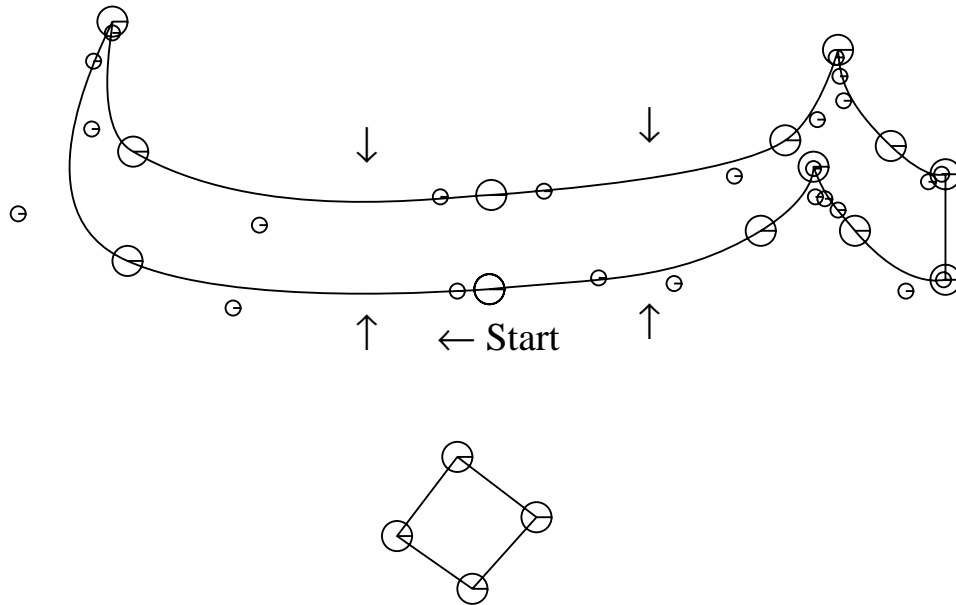


Figure 11: Plot of outline of connecting-previous baa.

A larger sized version of these same characters are shown in Figure 5 of the Srouji and Berry paper.

The next form considered was the stand-alone form of the *qaf*. This proved to be a case of stretching across the middle of a single Bézier curve. Figure 12 shows the outline of the form in a form similar to that of Figure 11. Only the start of the path that has to be stretched is marked.

The defined character at point size 30 and a 1 emm stretching of it are:

ق ق

Once it was clear how to proceed, the author got the assistance of Asaf Segal, a student needing additional academic credits, to carry out the stretching on the entire font. For the most part, adding stretching to the stretchable forms went quite straightforwardly after seeing the path followed by the outline for the form. However, one form, the stand-alone *alef maksura*, proved to defy an æsthetic stretching. Items 1, 3, 4, and 5 of Figure 13 shows various different attempts. The outline of this form had a strange situation that had never appeared before. In all but this form, the upper and lower strokes surrounding the stretched portions were implemented as the same number of 4-point curves. For example, in the connecting-previous *baa*, both the upper and lower strokes consisted of two curves and in the stand-alone *qaf*, both the upper and lower strokes consists of one curve. Therefore in both of these cases, the method of stretching both strokes is the same. In the

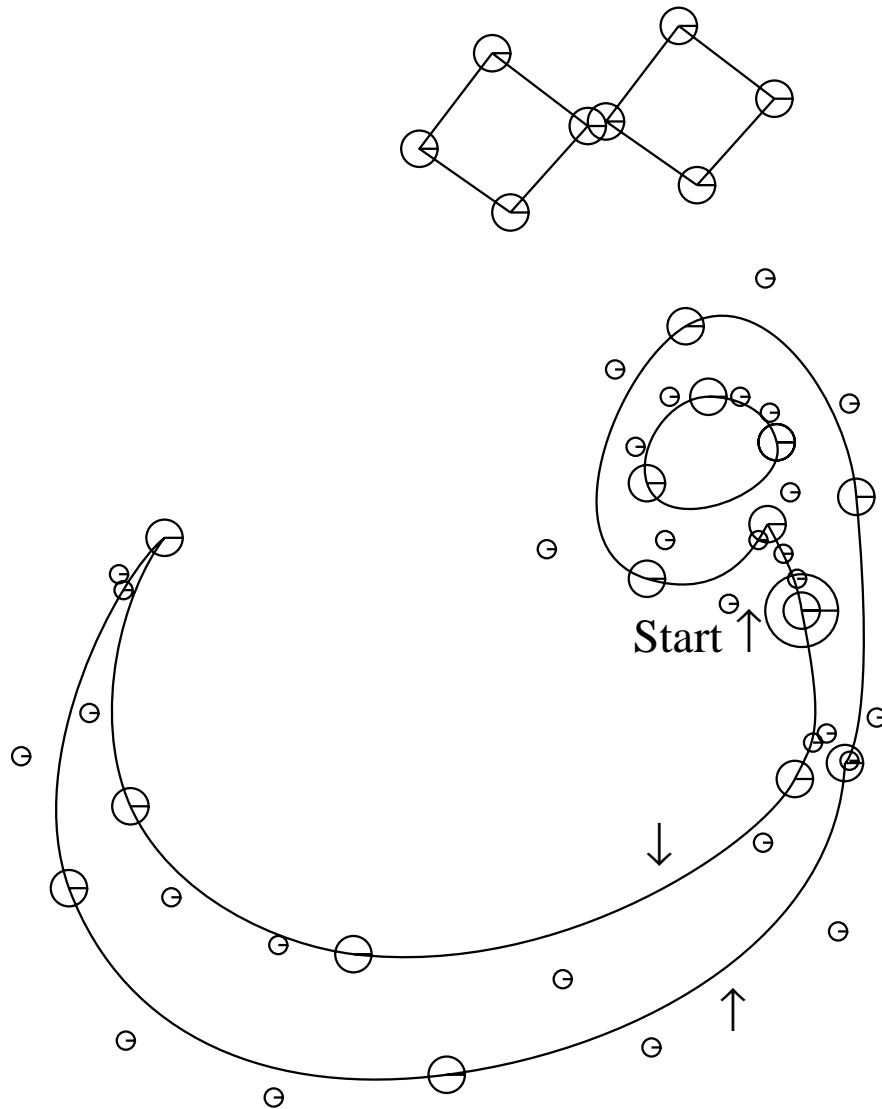


Figure 12: Plot of outline of stand-alone qaf.

stand-alone alef maksura, the lower stroke consists of one curve while the upper curve consists of two curves. It appeared that no matter how stretch in these curves was divided, the results were ugly.

After I had failed to get a pleasing stretch for the stand-alone alef maksura, I recalled Don Knuth's difficulties with getting a pleasing letter S when he was building the first version of the computer modern fonts with his then new METAFONT system. Interestingly,

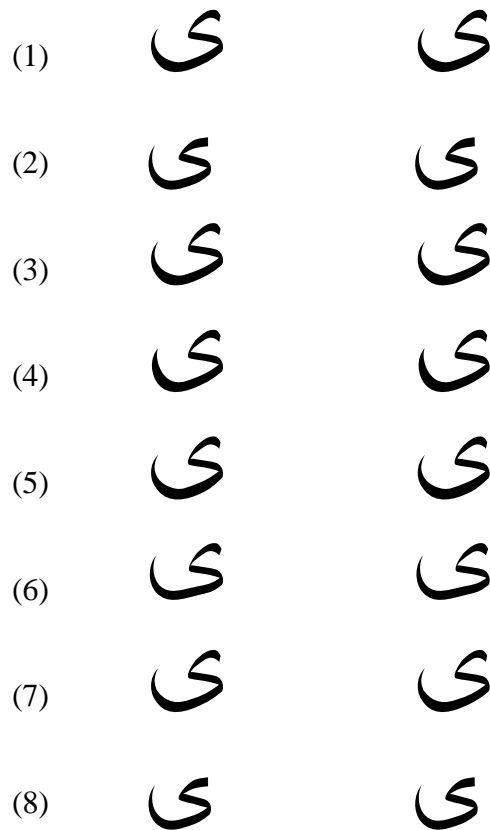


Figure 13: Various attempts to stretch the stand-alone alef maksura.

in a fashion, the alef maksura can be considered S shaped. After failing many times, he even considered writing his next book without the letter S. He remarked that the most important words, “Donald E. Knuth” did not require a letter S. Well, I too despaired of producing a stretchable stand-alone *alef maksura* and considered writing this paper without a stretched stand-alone alef maksura, but in my case, the most important words “Daniel Berry” requires a stand-alone *yaa* at the stretchable end of the second word, and the *yaa* is an *alef maksura* with two dots added underneath. So I *had* to find a solution. Knuth finally found a solution when his wife suggested making the S S-shaped. When I showed the problem to my wife, she did not suggest anything. Moreover, the obvious dictum of making the stand-alone *alef maksura* stand-alone-*alef-maksura*-shaped did not help me think of any solution.

Realizing the hopelessness of trying to get along without the stand-alone *alef maksura* and its relatives, I cheated! I had noticed, as shown in item 2 of Figure 13, that the connecting-before *alef maksura* stretched just fine. So I built a new stand-alone *alef maksura* from the bottom portion of the connecting-before *alef maksura* and the top portion of the stand-alone *alef maksura*. This construction is item 6 of the same figure. This version stretched properly, but the new letter did not go as deeply below the baseline as it

should. In the mean time, I got another student, Yaniv Bejerano, also needing extra academic credit points, to work on the problem. His final solution was to change the outline of the form so that the upper stroke consists of only one four-point Bézier curve. It proved impossible to duplicate with one fewer curve the original outline, but by fine adjustment of the control points, Bejerano managed to come very close, closer than can be discerned by the human eye at even phototypesetter resolution. Furthermore, when we showed the results to a small sample of people here no one could see the difference at the usual point sizes. Figure 14 shows the old and new outline side by side and below that superimposed; note that the words “Old” and “New” overlap. The reader can see, in the superimposed outlines, that the upper strokes of the bottom portion is slightly different.

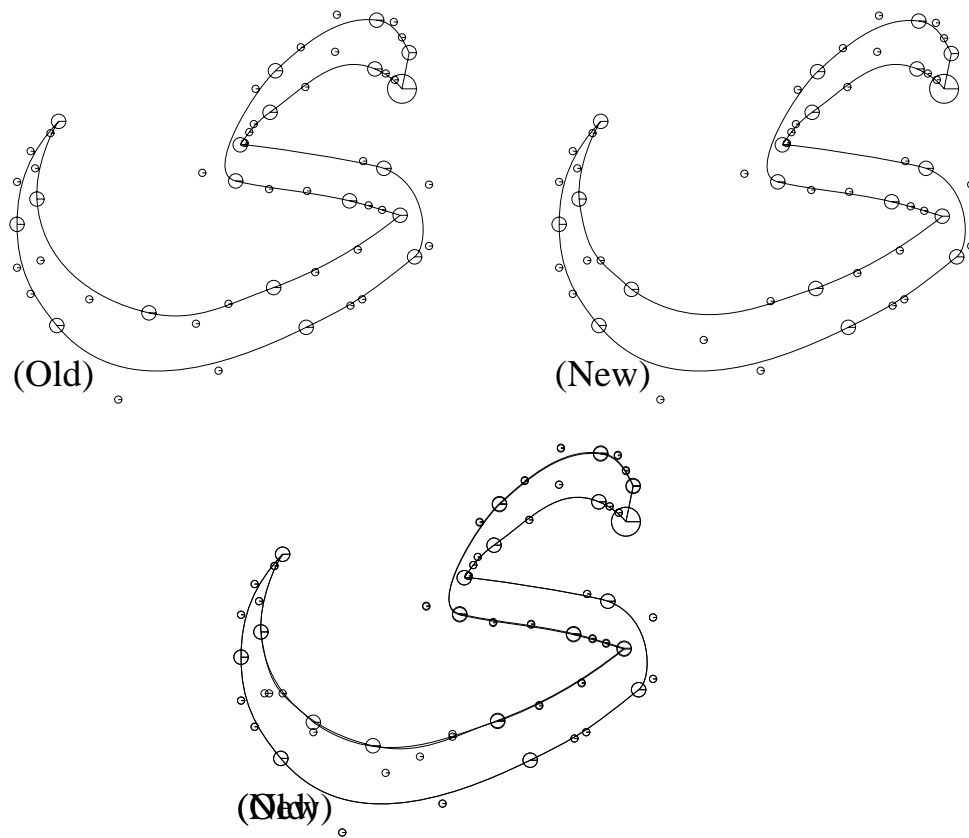


Figure 14: Old and new alef maksuras individually and superimposed.

Item 7 of Figure 13 shows the final form adopted and a stretched version. As a side-effect to the investigation, Bejerano found an improved way to stretch the connecting-before form, as shown in item 8 of the same figure.

7.2 Stretching Connections

Once it is understood how to stretch letters themselves with dynamic fonts, the same ap-

proach can be used to provide alternative, improved and more æsthetic methods to stretch connections, with and without the baseline filler. First, by making the filler a stretchable letter with a normal width of zero, it is possible to get smaller connection stretches than the size of one fixed-length filler. This makes it possible to avoid the problem of what to do when the needed stretch is not an integral multiple of the length of the fixed-length filler. Second, by applying the letter stretching technique directly to the connecting parts of connecting letters, it is possible to avoid using the filler entirely. The result is a smoother connection stretch; gone are the long flatness and the sudden corners where the filler or fillers meet the connecting parts of its neighbors. In their place is a gently sweeping curve with a slope of zero only momentarily where the two connecting parts connect. Figure 15 shows the connection between the connecting after *baa* and the connecting before *baa* first done with stretching the connecting parts and then with a filler. The amount of stretching is the same difference accentuating amount in both cases.

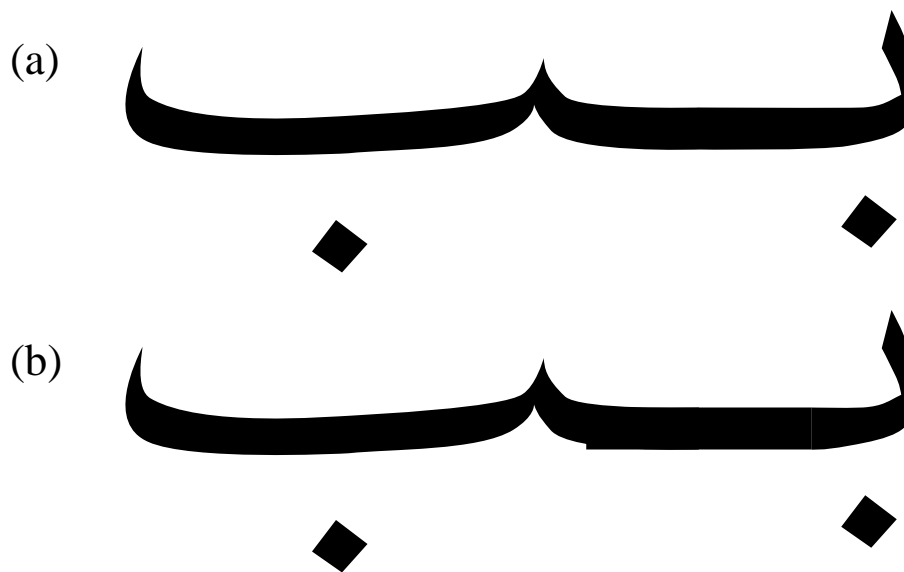


Figure 15: Stretchable connecting parts (a) vs. fillers (b).

In the POSTSCRIPT outlines, it is much easier to find the connecting parts of letters than the parts of letters themselves to be stretched, and once the connecting parts are found, it is easier to modify them for stretching than the parts of letters themselves to be stretched. Each connecting part consists of a four-point Bézier curve arriving at a vertical line segment, the vertical line segment, and a four-point Bézier curve leaving the vertical line segment; moreover, all such vertical line segments are exactly the same length since they have to be placed adjacent to each other. In the Arabic font used in this paper, each such line segment was either `0 56 rlineto` or `0 -56 rlineto`. Still moreover, the Bézier curves going in and out of the line segment have a slope of zero at the corner with the line segment. It is straightforward to stretch these zero-sloped Bézier curves in the method suggested by Figure 10.

8 MODIFICATIONS TO `ffortid`

The major changes required to `ffortid` are

1. the addition of the possibility of stretchable letters when replacing justification by stretching,
2. the addition of the possibility to stretch connections via a single stretchable filler of zero length rather than via sequences of fixed-length fillers,
3. the addition of the possibility to stretch connections by stretching the connecting parts of letters, say by putting half of the stretch amount into the connecting part of the before letter and the other half into the connecting part of the after letter,
4. the addition of some more command line options to control the nature of stretching,
5. the addition of some new fields to the width tables shared by `ditroff` and `ffortid`.
6. the addition of recursive reversing to deal with embedded X text inside Y text inside a X document, where X and Y are opposite directions of flow.

The description of the changes is driven by explaining the new command line options and the new width table fields. The changes to the algorithm is apparent in this discussion.

The command line to invoke `ffortid` is of the form:

```
ffortid [-rfont-position-list] ... [-paperwidth] [--font-position-list] ...
        [-s[n][[l|c|e|b]][f|2|m[amount]|a|ad|al]] [-ms[c|l|w] ...
```

The job of `ffortid` is to take the `ditroff` output which is formatted strictly left-to-right, to find occurrences of text in a right-to-left font and to rearrange each line so that the text in each font is written in its proper direction.

In the command line, the `-rfont-position-list` argument is used to specify which font positions are to be considered right-to-left.

In addition to the specified font directions, the results of `ffortid`'s reformatting also depends on the document's *current formatting direction*, which can be either left-to-right or right-to-left. The default formatting direction is left-to-right and can be changed by the user at any point in the document through the use of the

```
\X' PL'
```

and

```
\X' PR'
```

escapes in the `ditroff` input.

If the current formatting direction is left-to-right, all formatting, filling, adjusting, indenting, etc. is to appear as occurring from left to right. In each `ditroff` output line, any sequence of contiguous right-to-left font characters is reversed in place.

If the current formatting direction is right-to-left, all formatting, filling, adjusting, indenting, etc. is to appear as occurring from right to left. Each `ditroff` output line is reversed, including both the left and right margins. Then, any sequence of contiguous left-to-right font characters is reversed in place.

Sometimes right-to-left text contains some embedded left-to-right text, such as a

street address in Hebrew contains a numeral using traditional western digits with the most significant digit to the left. If this right-to-left text were embedded inside left-to-right text, e.g., an English sentence announcing a Hebrew street address, inside a left-to-right document, then the numeral, being left-to-right text, would be treated as left-to-right text that separates two right-to-left chunks inside a left-to-right document and the two chunks would end up flowing from left to right rather than being a single right-to-left chunk. The same problem occurs in the opposite directions as well. Let X and Y be two opposite directions. The user can announce embedded X text inside Y text in order to insure that its visual ordering is what is intended even when inside a X document by surrounding the embedded X text with the

`\X' {'`

and

`\X' }'`

escapes. The time-ordered input that produces the correct street address example of Section 4 is (with each input character shown in its output font).

He lives at `\f(hDהנהשיא \fR\X' { '49\X' } '\f(hDאב\ fR\X' { '15\X' } '\f(hDחיה`
`\fRin Israel.`

The `-paperwidth` argument is used to specify the width of the paper, in inches, on which the document will be printed. `ffortid` uses the specified paper width to determine the width of the right margin. The default paper width is 8.5 inches and like the font directions, it remains in effect throughout the entire document.

The `--font-position-list` argument is used to indicate which fonts positions, generally a subset of those designated as right-to-left, contain fonts for Arabic, Hebrew, Persian, or related languages. For these fonts, left and right justification of a line can be achieved by stretching instead of inserting extra white space between the words in the line. If requested by use of the `-s` argument described below, stretching is done on a line only if the line contains at least one word in a `--` designated font. If so, stretching is used in place of the normal distributed extra white space insertion for the entire line. The intention is that stretching soak up all the excess white space inserted by `ditroff` to adjust the line. If there are no opportunities for stretching or there are too few to soak up all the excess white space, what is not soaked up is distributed in between the words according to `ditroff`'s algorithm. There are several kinds of stretching, and which is in effect for all `--` designated fonts is specified with the `-s` argument, described below. If it is desired not to stretch a particular Arabic, Hebrew, Persian, or other font, while still stretching others, then the particular font should not be listed in the `--font-position-list`. Words in such fonts will not be stretched and will be spread with extra white space if the containing line is spread with extra white space.

The `-r` and the `--` specifications are independent. If a font is in the `--font-position-list` but not in the `-rfont-position-list`, then its text will be stretched but not reversed. This independence can be used to advantage when it is necessary to designate a particular Arabic, Hebrew, Persian, or other font as left-to-right for examples or to get around prob-

lems in the use of `eqn`, `ideal`, `pic`, or `tbl`.

The kind of stretching to be done for all fonts designated in the `--font-position-list` is indicated by the `-s` argument. There are two relatively independent dimensions that must be set to describe the stretching, what is stretched and the places that are stretched. A stretch argument is of the form

`-smp`

or

`-sn`

where m specifies the stretching mode, i.e, what is stretched, and p specifies the places that are stretched. The m and p must be given in that order and with no intervening spaces. The `-sn` means that there is *no* stretching and normal spreading of words is used even in `--` designated fonts. The choices for the mode m are:

1. `l` (letter ell): In the words designated by the p , stretch the last stretchable letter.
2. `c`: In the words designated by the p , stretch the last connection to a letter.
3. `e`: In the words designated by the p , stretch either the last stretchable letter or the last connection to a letter, whichever comes later.
4. `b`: In the words designated by the p , stretch either the last stretchable letter or the last connection to a letter, whichever comes later, and if it is a letter that is stretched and it is a connect-previous letter then also stretch the connection to the letter.

Not all modes are available for all fonts. For example, fonts for Hebrew, whose letters are not connected do not support connection stretching. While Arabic, Hebrew, and Persian traditionally do have letter stretching, not all fonts for them support letter stretching. `ffortid` attempts to stretch all `--` designated fonts in the specified modes, but in any text, ends up doing only those stretches that are possible given in the text's current font. To allow `ffortid` to know what stretches are possible, the width tables for stretchable fonts have some additional lines that must come somewhere after the name line and before the `charset` line.

```
stretchable: letters connections
stretchable: connections letters
stretchable: connections
stretchable: letters
```

Each such stretchable font must have one of the first four lines. We now discuss the various ways that different kinds of stretch are achieved in the available fonts and how `ffortid` deals with each.

To our knowledge, all Arabic and Persian fonts, have a base-line filler that can be used to achieve the stretching of connections. It is fairly easy for such a filler to be added to any font definition that does not have it, and moreover to make it the character that is addressed by `\(hy`, which is normally the code for the hyphen character. Since Arabic and Persian do not have a hyphen and hyphenation is turned off when in an Arabic or Persian font, it is safe to use `\(hy` to name the filler. Of course, this requires that the

width table for Arabic and Persian fonts have an entry for `hy` designating the filler character in the font, for example:

```
hy      15 0 0267      filler
```

Giving the filler character an explicit `ditroff` two-character name allows `ditroff` to deal with it uniformly despite that it might be in a different position in each font.

On the other hand, stretching of letters requires a dynamic font which, by its very nature of not having a constant bitmap for a given font name, point size, and character name, cannot be type 1, in `POSTSCRIPT` terminology, and cannot be a bitmapped font. Therefore, not all Arabic and Persian fonts support stretching of letters. Moreover, within a dynamic font, not all characters are stretchable. Historically, only characters with strong horizontal components are stretchable, such as those in the stand-alone and connect-previous forms of the *baa* family. Obviously, one cannot stretch totally vertical characters such as *alif*. Therefore, it is necessary to specify by additional information in the `ditroff` width table for a font which characters are stretchable. In the width table for an Arabic or Persian font, for each character that is not also an ASCII character, i.e., not also a digit or punctuation, and thus is neither connected or stretchable, one specifies after the name, width, ascender-descender information, and code, two additional fields, the connectivity and the stretchability of the character, in that order. The connectivity is either

```
N      for stand alone,
A      for connect after,
P      for connect previous,
B      for connect both, or
U      for unconnected, because it is punctuation or a diacritical, etc.,
```

and the stretchability is either

```
N      for not stretchable,
S      for stretchable,
```

Some examples of width table lines are:

```
%%      125 2 045      percent
---     55 0 0101    U      N      comma
---     70 0 0105    U      N      hamza
---     129 0 0106   N      S      baa_SA
---     36 2 0102    N      N      alef_SA
---     113 0 0177   A      N      sad_CA
---     66 2 0215    A      S      caf_CA
---     43 2 0225    P      N      alef_CP
```

```

--- 120 0 0274 P S baa_CP
--- 53 0 0230 B N baa_CB
--- 73 2 0261 B S caf_CB

```

Recall that --- in the name field of a character means that it can be addressed only by `\N'n'`, where n is the decimal equivalent of the character's code. Only such lines will have the connectivity and stretchability fields. Also in the character name which is a comment in the table, `_SA` means "stand alone", `_CA` means "connecting after", `_CP` means "connecting previous or before", and `_CB` means "connecting both".

For a Hebrew font, for which there is no notion of connectivity of letters, and therefore, the position of the letters is irrelevant for deciding stretching, there is only the possibility of stretching letters. Some examples of width table lines for such fonts are:

```

%      132 3 045   percent
--- 95 3 0140 U N   quoteleft=alef
--- 92 3 0141 U S   a=bet

```

In a dynamic font, there are two additional, alternative ways that stretching of connections can be achieved.

- The filler is a stretchable letter, normally of width zero, to which the total width of the filler is passed as the stretch amount.
- The connecting portions of all connecting letters are themselves stretchable in the same way as the stretchable letters are. In this situation to achieve a total connection stretch of x , one would pass $x/2$ to each of the connecting-after portion of the before letter and the connecting-before portion of the after letter.

The use of the first of these solves the problems caused by the fact that amount of a given connection stretch may not be integrally divisible by the width of the filler. A stretchable filler can be stretched to any amount. The use of the second improves the appearance of the connection stretch. While letter stretching is done with nice, smooth curves, connection stretching using the very straight filler is noticeably flatter and there are observable corners where the filler meets the generally curved connecting parts of its adjacent letters. While the fixed-size filler seems to be available on all Arabic and Persian fonts, stretchable fillers and stretchable connecting parts are available only with type 3 POSTSCRIPT fonts, although it would be possible to provide a stretchable filler as the only locally defined character in a type 3 font that falls to another type 1 font for all the other characters, which are only virtual in the type 3 font.

The `ditroff` width table for any font providing a stretchable filler or stretchable connecting parts must have an additional line to specify the nature of the connection stretch in the font, which must be one of the following.

```

connection stretch: fixed filler
connection stretch: stretchable filler
connection stretch: stretchable connections

```

This line must come somewhere after the `name` line and before the `charset` line. If none is specified, it is assumed to be the first. Therefore, it is not necessary to say anything new for the typical type 1 or bitmapped font with a fixed-sized filler. Note that if a font allows different kinds of connection stretching, only one can be specified per mounting of the font specified in a single width table. If one wants to use the same font with different ways of stretching connections, one must mount the same font under different names in different width tables, each specifying a different kind of connection stretching.

`ffortid` implements the connection stretching that is requested by the `-s` command-line argument as well as it can using the kind of connection stretching available for the font being used. Thus, if one is not using fixed-sized fillers, `ffortid` ignores the various options put in to deal with the fact that an integral number of fillers may not fulfill the needed stretch.

Below, “stretchable unit” refers to what is a candidate for stretching according to the mode. The choices for p , which specifies places of stretching, are:

1. `f`: In any line, stretch the last stretchable unit.
2. `2`: Assuming that the mode is `b` (both), in any line, stretch the last two stretchable units, if they are the connection leading to a stretchable connect-previous letter and that letter, and stretch only the last stretchable unit otherwise. If the mode is not `b`, then this choice of places is illegal.
3. `nm` or `m`: In any line, stretch the last stretchable unit by an amount not exceeding n emms. If that does not exhaust the available white space, then stretch the next last stretchable unit by an amount not exceeding n emms, and so on until all the available white space is exhausted. If n is not given, it is assumed to be 2.0 . In general, n can be any number in floating point format.
4. `a`, `ad`, or `a1`: In any line, stretch all stretchable units. In this case, the total amount available for stretching is divided evenly over all stretchable units on the line identified according to the mode. Since the units of stretching are the units of device resolution, the amount available might not divide evenly over the number of places. Therefore, it is useful to be able to specify what to do with the remainder of this division. This specification is given as an extension of the stretching argument. The choices are `d` or `l`, with the former indicating that the excess be distributed as evenly as possible to the spaces between words and the latter indicating that the excess be distributed as evenly as possible in stretchable letters only. The latter is the default if no choice is specified.

Sometimes, it is desirable to be able to manually stretch connections or letters to achieve special effects, e.g., more balanced stretching or stretching in lines that are not otherwise adjusted, e.g., centered lines. Stretching a connection can be achieved by using the base-line filler character explicitly as many times as necessary to achieve the desired length. Note that the `ditroff` line drawing function can be used to get a series of adjacent fillers to any desired length, e.g.,

```
\l'2m\ (hy'
```

will draw a string of adjacent base-line fillers of length 2 emms.

To achieve stretching of letters, one should immediately precede, with no intervening white space, the letter to be stretched by

```
\X'stretch'\h'n'
```

where n is a valid length expression in `ditroff`'s input language. `ffortid` is prepared to deal with the output from `ditroff` generated by this input to generate output that will cause the letter immediately following it to be stretched by the length specified in n . For example,

```
\X'stretch'\h'1m'\N'70''
```

will cause the character whose decimal code is 70 to be stretched by 1 emm. The output will fail to have the desired effect if the letter following is not a stretchable letter.

If connection stretching is achieved by having a stretching filler, then one manually stretches the filler character by the desired amount as if it were a letter.

```
\X'stretch'\h'n'\(hy
```

Here, though the stretch parameter n is the total length of the filler, as the filler is of length zero if it is not stretched.

To stretch the connecting parts of letters, two additional escape sequences are provided that may be placed before, with no intervening printable text, the letter to which they apply,

```
\X'BCstretch'\h'nb'
```

```
\X'ACstretch'\h'na'
```

where nb and na are valid length expressions in `troff`'s input language. These specify the amounts of stretch in the before and after connecting parts of the immediately following letter. The order in which the `\X'stretch'\h'n'`, `\X'BCstretch'\h'nb'`, and `\X'ACstretch'\h'na'` for a letter appear is irrelevant, but in between them and after the last of them, there is *no* printable text, including white space (including new lines), and the letter to which they apply immediately follows the last. Suppose that two consecutive, in logical order, letters have decimal codes 70 and 80. Suppose also that 70 connects after to the connecting before 80. Suppose finally that this connection from 70 to 80 is to be stretched by 1 emm and the letter 80 is to be stretched by 2 emms. Then the input would look as follows (but not split into two lines, in fact with no internal white space at all):

```
\X'ACstretch'\h'.5m'\N'70'\X'BCstretch'\h'.5m'
```

```
\X'stretch'\h'2m'\N'80'
```

Note that the connection stretch of 1 emm was split into two stretches of .5 emm for each of the connecting after and the connecting before parts.

For finer control over stretching, it may be desirable to inhibit automatic stretching on manually stretched connections and letters. Accordingly, two command line flags are provided for this purpose:

1. `-msc`: Do not automatically stretch manually stretched connections.
2. `-msl`: Do not automatically stretch manually stretched letters.

3. $-msw$: Do not automatically stretch any word containing any manual stretching.

9 EXAMPLES

All of these samples use a very narrow column width in order to exaggerate the differences and to force whatever stretching that is done to be very pronounced. In all the samples, the last line has no stretching since it is not full and in the normal word-spreading realm, no additional space would appear between the words. Because only one setting of the stretch option applies for the whole of a given document, all the samples were typeset as other documents and were included in this paper as encapsulated POSTSCRIPT figures. Finally, each example is designated by its language and the stretch option that was used to typeset it.

In Hebrew, letters are not connected. Therefore, only letters are stretchable, and the width tables reflect this fact. Therefore, all $-scPs$ for any place P , look like $-sn$, each $-sbP$ and $-seP$ looks like the corresponding $-slP$, with the special case of $-sb2$ looking like $-slf$. Therefore, only the $-sla$ and the $-slf$ examples are shown.

הַצִּיּוֹךְ בְּדִיקָה עֲצֵמִית, מֵאֲמֵר זֶה
 נִסְזוּר בְּאֲמִצְעוֹת הַמַּעֲכֵכִים ת
 שְׂתוֹדֵא הַלְעִיל, וְהוּא כּוֹל ל
 דּוֹגְמָאוֹת חֲבוֹת שֶׁל כֵּת ב
 בַּעֲבִיבִית, וְעַבִּיבִית, חֲפֵסִית.

Figure 16: Hebrew with $-sla$.

הַצִּיּוֹךְ בְּדִיקָה עֲצֵמִית, מֵאֲמֵר זֶה
 תִּסְזוּר בְּאֲמִצְעוֹת הַמַּעֲכֵכִים
 לִשְׂתוֹדֵא הַלְעִיל, וְהוּא כּוֹל
 בִּדּוֹגְמָאוֹת חֲבוֹת שֶׁל כֵּת
 בַּעֲבִיבִית, וְעַבִּיבִית, חֲפֵסִית.

Figure 17: Hebrew with $-slf$.

In Arabic, both letters and connections are stretched. The first group of examples, the $-sMfs$, show stretching in final places in a line and the second group, the $-sMas$ shows stretching in all places on a line. As expected, the stretching is more pronounced in the $-sMfs$. For $-scf$, in all non-last lines, the connections absorb all of the stretch. The effect is most pronounced in lines 2 and 3.

كإختبار ذاتي للبرنامج،
 هذه المقالة طُبِعَتْ
 بواسطة البرنامج
 الموصوف أعلاه، وتحتوي
 على عدة أمثال لِنُصِّص
 كُتِبَت باللغات العربية،
 العبرية، والفارسية.

Figure 18: Arabic with *-scf*.

For *-slf*, the final stretchable letter of each line absorbs all the stretch, with the effect being most pronounced in line 2. In lines 1 and 6, the last stretchable letter is not the last letter in the line. In line 3, there is no stretchable letter at all, so the words are left spread apart.

كإختبار ذاتي للبرنامج،
 هذه المقالة طُبِعَتْ
 بواسطة البرنامج
 الموصوف أعلاه، وتحتوي
 على عدة أمثال لِنُصِّص
 كُتِبَت باللغات العربية،
 العبرية، والفارسية.

Figure 19: Arabic with *-slf*.

For *-sef*, the final stretchable letter or connection, whichever is later (left most), and not both, gets stretched. The effect is most pronounced in lines 2 and 3. In line 2, the last letter is stretchable so it gets all the stretch. In line 3, the final letter is not stretchable, but it is connected to the previous letter, so that connection absorbs all the stretch.

كإختبار ذاتي للبرنامج،
 هذه المقلّة طَبَعَتْ
 بواسطة البرنامج م ج
 الموصوف أعلاه، وتحتوي
 على عدة أمثال لِنُصِّصْ
 كُتِبَتْ باللغّات العربيّة،
 العبريّة، والفارسيّة.

Figure 20: Arabic with *-sef*.

The *-sb2* option is similar to the *-sef* option, except that when a letter would be stretched and that letter is connected, then the letter and its connection split all the stretch. In line 2, each of the main body and the connecting part of the final *taa* absorbs half of the total stretch. In line 3, the final letter is not stretchable, but it is connected; so, that connection absorbs all the stretch. In line 4, the final letter is stretchable, but it is not connected; therefore the letter absorbs all of the stretch.

كإختبار ذاتي للبرنامج،
 هذه المقلّة طَبَعَتْ
 بواسطة البرنامج م ج
 الموصوف أعلاه، وتحتوي
 على عدة أمثال لِنُصِّصْ
 كُتِبَتْ باللغّات العربيّة،
 العبريّة، والفارسيّة.

Figure 21: Arabic with *-sb2*.

For the *-sMas*, the total stretch is distributed over all candidates according to the *M*. Thus, each individual stretch tends to be shorter than for the *-sMfs*. The *-sca* option causes only the and all the connections to be stretched.

كاختبار ذاتي للبرنامج،
 هذه المقالة طُبِعَت
 بواسطة البرنامج
 الموصوف أعلاه، وتحتوي
 على عدة أمثال لنُصِّص
 كُتِبَت باللغات العربية،
 العبرية، والفارسية.

Figure 22: Arabic with *-sca*.

For *-sla*, it is expected that all stretchable letters in a line be stretched the same amount. For all lines, except lines 2 and 3, the stretches are hardly noticeable since each stretch is a fraction of the total stretch for the line. Line 2 has only one stretchable letter, and it absorbs all the stretch. Line 3 has no stretchable letter; therefore, the words remain spread with additional white space.

كاختبار ذاتي للبرنامج،
 هذه المقالة طُبِعَت
 بواسطة البرنامج
 الموصوف أعلاه، وتحتوي
 على عدة أمثال لنُصِّص
 كُتِبَت باللغات العربية،
 العبرية، والفارسية.

Figure 23: Arabic with *-sla*.

For *-sea*, each word has at most one stretch, which is either in the body of the last stretchable letter or in the last connection, which ever is later in the word. In line 2 and 3, there are few enough such places that the stretch per place is quite pronounced.

كإختبار ذاتي للبرنامج،
 هذه المقالة طُبِعَت
 بواسطة البرنامج
 الموصوف أعلاه، وتحتوي
 على عدة أمثال لنُصص
 كُتبت باللغات العربية،
 العبرية، والفارسية.

Figure 24: Arabic with *-sea*.

The *-sba* case differs from the *-sea* case in line 2. Both the last letter and its connection are stretched the same amount as the other stretchable places in the line, which just happen to be connections

كإختبار ذاتي للبرنامج،
 هذه المقالة طُبِعَت
 بواسطة البرنامج
 الموصوف أعلاه، وتحتوي
 على عدة أمثال لنُصص
 كُتبت باللغات العربية،
 العبرية، والفارسية.

Figure 25: Arabic with *-sba*.

For Persian, only examples with results differing significantly from the corresponding Arabic examples are shown. With the Persian text, the *-sb2* case yields more pronounced results. There are several occurrences of connecting before *ye* in lines 5, 6, and 7 that end up being the stretch places of their lines. Each of these lines shows that *ye* and its before connection stretched the same amount.

بعنوان يك خودآزمائی،
 این مقاله با استفاده از
 سیستمی که در اینجا
 تشریح میشود نقشبندی
 شده و مثالها د ی از
 نوشته‌های عربی،
 عبری، و فارسی در
 متن خود دارد.

Figure 26: Persian with *-sb2*.

In the *-sef*, *-scf*, and *-slf* examples, which are not shown, the same *yes* are stretched, but all the stretch is absorbed by the body of the letter, the connection, and the body of the letter, respectively. Among the *-sMas*, which are not particularly remarkable, only the *-sea* example is shown.

بعنوان يك خودآزمائی،
 این مقاله با استفاده از
 سیستمی که در اینجا
 تشریح میشود نقشبندی
 شده و مثالها د ی از
 نوشته‌های عربی،
 عبری، و فارسی در
 متن خود دارد.

Figure 27: Persian with *-sea*.

An imitation, in which each letter is stretched by one emm, of the caligraphic example of Figure 3 is shown below.

ب ف گ

This example shows that the balanced stretching can be achieved by manual control, but it also shows that the font being used is not very good for stretching. Now that the tech-

nique to do stretching has been demonstrated, it is necessary to design a font whose letters stretch more gracefully.

On the first page of the paper, the stand-alone *yaa* in the author's Arabic name and the *resh* in the author's Hebrew name were stretched manually for the purpose of making each family name the same length as its corresponding private name. The amount of stretch was calculated using the ditroff internal length functions to put into a register the difference in the lengths of the family and private names.

10 SLANTED BASELINE WRITING

Examination of examples of Persian printing shows that

1. it uses fonts whose characters are observably different from those of fonts typically used for Arabic,
2. stretching of characters seems to be more prevalent than in Arabic printing almost to the exclusion of stretching of connections, and
3. words, but not lines seem to be written on a slanted baseline.

These observations are clear in Figure 28, which shows a sample of some Persian printing from the author's favorite Persian cookbook [15] and Figure 29, which shows a sample of some Persian calligraphy.

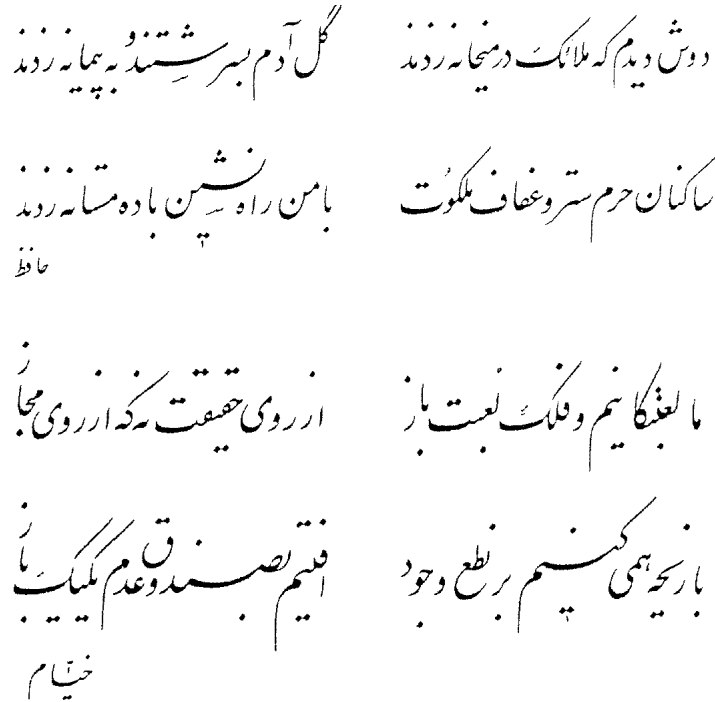


Figure 28: Persian printing.

The font that is used in the cookbook excerpt is called Nastaliq. To date, neither the author nor any others contributing to an Internet Arabic Script group have seen a decent



Figure 29: Persian calligraphy.

POSTSCRIPT outline font for Nasta'liq or any other font with the distinctly Persian flavor. As a consequence, at present, computer-aided Persian typesetting seems to be done using Arabic fonts for results that are not entirely æsthetic to the Persian eye.

First, by making copies of a large variety of samples and marking these copies up with lines showing the flow of words and lines running through the slanted baselines, it was determined that while the line flows horizontally, the baseline of individual words or individual groups of consecutive short words is generally slanted 22° downward from the horizontal. (Recall that Persian text flows from right to left, so the slant is described in these terms.) The sequence of the center points of the slanted-word-or-group-of-word baselines for a single line of text describe a single horizontal line segment, which is taken as the axis or baseline of the whole line of text. Figure 30 shows a schematic of the baselines of words or groups of words that form a single line.

Furthermore, it appears that the beginning of any but the first word on a line is im-

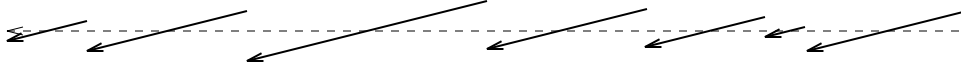


Figure 30: The slanted baselines of words in one line.

mediately above the end of the previous word. That is, from the point of view of projections of the words onto the horizontal axis of the line, there is no spacing between words. The separation of the words is achieved by the vertical clearance between the end of one word and the beginning of the next right above it. In the above, when a group of consecutive short words is treated as a unit for slanting, there is some horizontally visible white space between the words. It appears that merging short words into a larger unit for slanting is a physical necessity as there would not be enough vertical clearance between one short word and the next if each were slanted separately. Finally, it appears that stretching is applied mainly to letters and not to connections.

It is this author's opinion that the reason no one has invested the considerable effort to produce a suitable Nastaliq font is that it was not understood how to implement in POSTSCRIPT and use in formatters the slanted baseline aspect. This section remedies this problem by analyzing the slanted baseline requirement and showing how it can be implemented in POSTSCRIPT fonts and used by formatters to achieve a very Persian appearance. Since the author does not even have a flat version of Nastaliq available as a type 3 POSTSCRIPT font, the author applied the slanting to the Arabic Naskh font with stretchable forms to produce a font named Persian Naskh also with stretchable letters and he used this font with a modified `ffortid` that knows how to use a slanting font to achieve the proper baseline slanting. It is hoped that the availability of the technology to define and use slanted-baseline-printing fonts with stretchable forms will provide the incentive for a good typographer to develop a high quality Nastaliq POSTSCRIPT font definition.

First the method of slanting POSTSCRIPT fonts is described. Then the modifications to `ffortid` and to `ditroff` width tables to support the use of slanting fonts is described.

11 SLANTING FONTS

As mentioned, it was decided to apply slanting to a new font called Persian Naskh which is a copy of a stretchable Arabic Naskh except for the stretching. The simplest way to achieve the slanting in all characters is to modify the font matrix so that what was horizontal is now slanted 22° upward (POSTSCRIPT fonts view characters as being laid out from left to right; therefore the slant is considered in the opposite direction from for the normal right-to-left flow.) and what was vertical is still vertical. Thus the font matrix defining line was changed from

```
/FontMatrix [.001200 0 0 .001200 0 0] def
```

to

```
/FontMatrix [.001200 .0004495 0 .001200 0 0] def %%22 degrees
```

The value `.0004495` will be recognized as $\tan(22^\circ) \times 0.0012$ where `.0012` is both the x

and the y value from the matrix

```
[.001200 .0004495 0 .001200 0 0] .
```

Once this font is defined and set, one can print any word with a slanted baseline by moving to the position of the right end of the word, below the line's axis and then showing the word. Figure 31 shows a slanted printing of the word *salaam*.

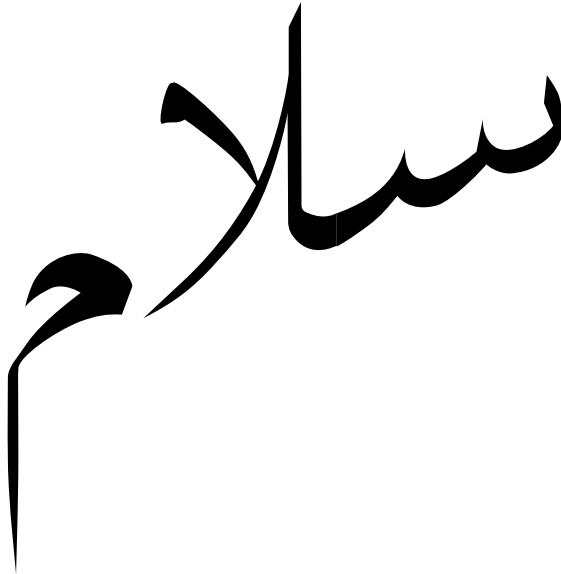


Figure 31: Slanted salaam.

This output was achieved by simply finding and setting the Persian-Naskh font and issuing a `show` command with the characters, from left to right, of the word *salaam*, as its string argument. The use of the font matrix to achieve the slanting insures that what the POSTSCRIPT `show` mechanism believes is a move in the x direction by the width of the last printed character is in fact a move upward and to the right. Figure 32 shows the example of Figure 5 of the Srouji and Berry paper printed using the Persian Naskh font. The purpose of this example is to show that both forms of stretching work quite handily in the presence of slanting.

12 MORE MODIFICATIONS TO `ffortid`

Again there were three main kinds of modifications to `ffortid`.

1. The layout algorithm of `ffortid` is changed so that it puts each word, after justification either by spreading words or stretching, into its own slanted baseline.
2. New command-line options are added.
3. The width tables shared by `ditroff` and `ffortid` are modified slightly.

These changes are discussed in reverse order.

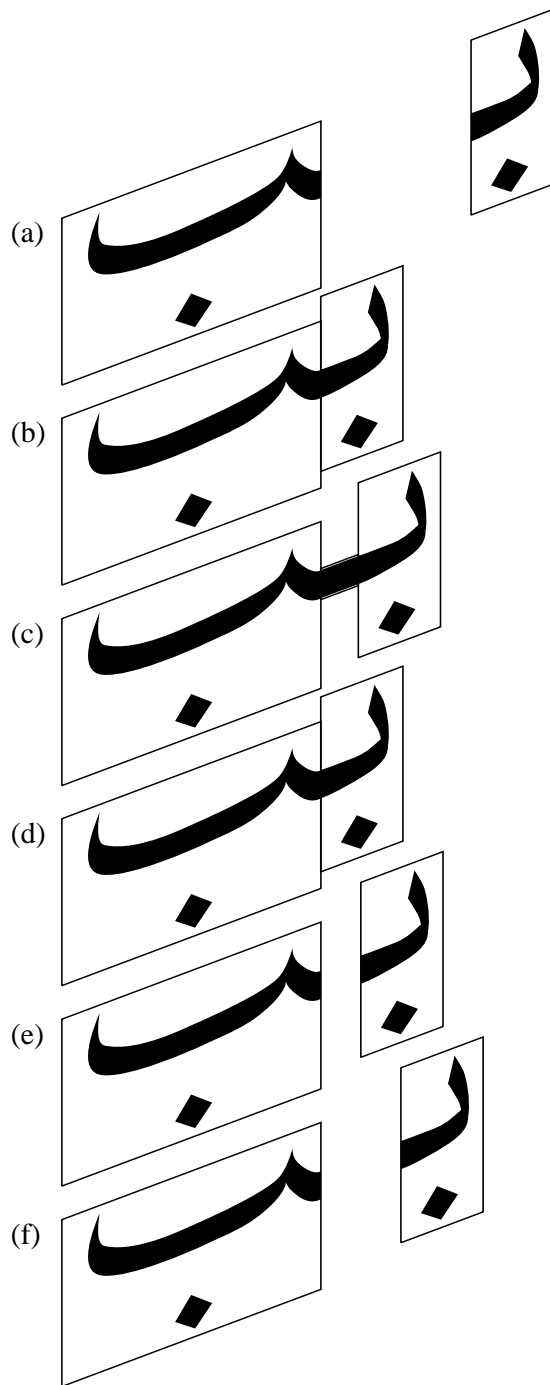


Figure 32: Non-stretched and stretched.

The width tables have to be modified slightly so that there is a line announcing that the font described by the table is slanted and that line gives the amount of slant in degrees, which must agree with the angle implicit in the font matrix of the font. In addition, in order to cause the beginning of one word to be just above the end of the previous with no horizontal movement, it is necessary to lie to `ditroff` and tell it that the normal interword space is 0, while providing another unpaddable space to be used explicitly by the input document to group short words together into a single slantable unit. First, `ditroff` would not accept a width of 0 for the space; it took it as the default width of $\frac{1}{3}$ emm. So, a space of width 1 was tried and it both was accepted by `ditroff` and produced visually acceptable results which the human eye could not distinguish from a space width of 0. Recall that the units of the space width is the unit of device resolution. Thus, for Adobe `transcript`'s `psc` device, a width of 1 is a width of $\frac{1}{576}$ inches. Figure 33 shows the second paragraph of the Persian abstract with the same spacing between the words but with nothing slanted. The reader can see that the words appear run together with no space between them. The unpaddable space was provided as the character named by the `ditroff` special character `\(ps`, for "permanent space" whose width is set to what was the width of the space before setting it to 1. The name for the unpaddable space had to be a two-character name that is mnemonic and is not in the special font; if it were in the special font, then it would not be possible to address the like-named special font character without explicitly mounting the special font.

بعنوان يك خود آزمايی، اين مقاله با استفاده از سيستمی كه در اينجا تشریح ميشود نقشبندي شده و مثالهایی از نوشته‌های عربي، عبری، و فارسی در متن خود دارد .

33: *Unslanted Persian text spaced as for slanted printing.*

The new command-line option is to indicate which font positions contain slanted fonts. Of course, it will not work to attempt, through the setting of this option, to slant a font whose width table does not say that it is slanted. However, the slanting algorithm, described in the next paragraph, does not happen to a font not in this list.

The change to the `ffortid` layout algorithm is that after all characters have been put in visual order and all justification and stretching requested by the user have been applied, the projection of the starting position of each word on its text line's axis is known. Note that the justification and stretching are affected by the fact that the space is of width 1. Of course, if stretching is not indicated, then as a result of justification, words may be more than 1 unit apart as a result of the additional white space inserted by the spreading of the words. If stretching is specified, then if there are stretchable places none of the interword gaps should be spread, as stretching should absorb all of the excess white space.

At this point `ffortid` calculates the width of each word and then inserts before the beginning of each word, of length `len_word`, a vertical movement to $v = \tan(\alpha) \times \frac{len_word}{2}$ units below the line's horizontal axis, where α is the angle of slant, usually 22° , determined from the font's width table. In order to avoid the line's axis drifting as a result of round-off error in digitizing the floating point tangent values, the vertical positioning before the printing of each word is accomplished by using the absolute vertical positioning instruction with an argument that is the sum of the `y` of the

line's axis and the calculated movement v for the word.

The Persian abstract of this paper was typeset in this slanted pseudo-Persian font. In preparing this example, it was learned that it is not a good idea to slant text that contains non-slantable text such as that in Latin letters. However, if one is going to slant such text anyway, then Persian text which is immediately adjacent to Latin text, such as parentheses and other punctuation or a single letter grammatical article or conjunction, should be in a non-slanting font whose appearance is coordinated to the slanting font. In this case, since the slanting Persian Naskh is a slanting version of Arabic Naskh, Arabic Naskh was used as the non-slanting font whose appearance is coordinated to that of Persian Naskh!

13 EXAMPLES

The same text used earlier for Persian examples is used for the slanted Persian examples. Only the $-slP$ examples are exhibited, because it appears that connection stretching is not used in slanted writing. The slanted Persian abstract at the beginning of the paper is typeset, as are all the abstracts, with $-sea$, and it shows connection stretching in slanted text.

بعنوان يك خود آزمائی، اين مقاله با استفاده
 از سيستم ي که در اينجا تشریح میشود
 نقشبندی شده و مثالهاي از نوشته‌ها
 عربی، هيس، و فارسی در متن خود دارد.

Figure 34: Slanted Persian with $-sla$.

بعنوان يك خودآزمائی، اين مقاله با استفاده
 از سيستمي و مثالهايي كه در اينجا تشریح میشود
 نقشهنگ شده و فارسي در متن خود دارد.

Figure 35: Slanted Persian with `-slf`.

In these examples, consecutive words are grouped together as single slanted units, in order that all slanted units in a line are approximately the same length. This way, no individual word extends far beyond the others and no individual word is so short that it does not have a clear separation from its neighbors. Between two consecutive words of a single slanted unit is an unpaddable space built into the slanted persian font, addressed by `\(ps` in `ditroff`. These unpaddable blanks are not candidates for stretching if there is nothing stretchable in the line and are not candidates for shrinking to give way to stretching. They insure horizontal separation of the words that make up a single slanted unit.

14 RELATED WORK

As far as this author has been able to determine, no other Arabic, Hebrew, or Persian formatting system provides letter stretching and slanting the baselines of words. This author is aware of Yannis Haralambous's `ScholarTEX` [16], Klaus Lagally's `ArabTEX` [17], and Don Knuth and Pierre MacKay's `TEX-XET` [18]. The closest that this author has seen are systems that provide multiple forms of stretchable forms that can be selected by the user. `TEX` systems typically use fonts generated by `METAFONT`, which are usually bitmaps. Bitmap-based formatters would be hard put to provide dynamically stretchable characters that are necessary to implement the solution outlined in this paper and implemented with `ditroff/ffortid`, a system that supports easy use of `POSTSCRIPT` outline fonts. There are options to use `POSTSCRIPT` outline fonts with `TEX`, but to date, only type 1 fonts seem to be available for the `METAFONT`-generated fonts. It would be necessary to obtain type 3 versions of these fonts so that the character definitions can be edited to make them dynamic.

15 CONCLUSIONS

This paper has described the addition of full stretching and slanting to Hebrew, Arabic, and Persian typesetting with `ditroff/ffortid`. The changes necessary to the `POSTSCRIPT` fonts and the `ffortid ditroff` postprocessor were described. As to whether the results are aesthetically pleasing is left to the reader. However, if the aesthetics are lacking, perhaps applying the techniques described in this paper to better designed fonts, perhaps designed specifically to be stretched and slanted, will yield more pleasing results. It is hoped that the proof of principle given in this paper increases the incentive for developing these better fonts.

ACKNOWLEDGEMENTS

The author thanks Jacques André, Bijan Arbab (بيژن ارباب), Farhad Arbab (فرهاد ارباب), Yaniv Begerano (יניב בגרינו), Avron Cohen (אברון כהן), Gershon Elber (גרשון אילבר), Achi Gvirtzman (אחי גבירצמן), Yannis Haralambous, Ziv Horesh (זיב חורש), Nir Katz (ניר כ"ץ), Don Knuth, Eli Leiba (אלי לייבה), Erez Manor (אריז מנור), Shahrzade Mazaher (شهرزاد مظا هر), Asaf Segal (אסף סגל), Johny Srouji (جونى سروجى), and Uri Yifrah (אורי יפרח) for their help and suggestions.

REFERENCES

1. J. Srouji and D.M. Berry, 'Arabic Formatting with `ditroff/ffortid`', *Electronic Publishing*, **5** (4), 163–208 (1992).
2. B.W. Kernighan, 'A Typesetter-independent TROFF', *Computing Science Technical Report No. 97*, Bell Laboratories (1982).
3. 'Arab Standards and Metrology Organization, 8-Bit Coded Arabic/Latin Character Set for Information Interchange', *ASMO DS 708*, Amman, Jordan (1985).
4. C. Buchman, D.M. Berry, and J. Gonczarowski, '`DITROFF/FFORTID`, An Adaptation of the UNIX `DITROFF` for Formatting Bi-Directional Text', *ACM Transactions on Office Information Systems*, **3** (4), 380–397 (1985).
5. مهدي السيد محمود، كيف تتعلم الخط العربي: نسخ، رقعة، ثلث، فارسي، مكتبة ابن سينا، للنشر والتوزيع والتصدير، القاهرة، مصر، ١٩٨٧.
6. Mahdi ElSayed Mahmud, *Learning Arabic Calligraphy: Naskh, Requah, Tholoth, Farsi, Ibn Sina*, Publisher, Cairo, Egypt, 1987.
7. 'Genesis (אִבְרָאָה), *Torah Scrolls (ספר יחזקיה)*, Found in any synagogue (Unknown).
8. *POSTSCRIPT Language Reference Manual, Second Edition*, Adobe Systems Incorporated, Addison-Wesley, Reading, MA, 1992.
9. D. Weise and D. Adler, 'TrueType and Microsoft Windows Version 3.1', *Technical Report*, Microsoft Corporation, Redmond, WA (1992).
10. J. André and B. Borghi, 'Dynamic Fonts', *POSTSCRIPT Language Journal*, **2** (3), 4–6 (1990).
11. 'Adobe Type 1 Font Format', *Part No. LPS0064*, Adobe Systems, Inc. (1990).
12. J. André and I. Vatton, 'Dynamic Optical Scaling and Variable Sized Characters', *Electronic Publishing—Origination, Dissemination, and Design*, **7** (4), 231–250 (1994).
13. J.D. Becker, 'Multilingual Word Processing', *Scientific American*, **251** (1), 96–107 (1984).
14. J.D. Becker, 'Arabic Word Processing', *Communications of the ACM*, **30** (7), 600–611 (1987).

-
15. N. Batmanglij, *Food of Life*, Mage, Washington, DC, 1986.
 16. Y. Haralambous, *Scholar T_EX*, Y. Haralambous, Lille, France, 1991.
 17. K. Lagally, 'ArabT_EX, a System for Typesetting Arabic, User Manual Version 3.00', *Report Nr. 1993/11*, Fakultät Informatik, Universität Stuttgart, Stuttgart, Germany (1993).
 18. D.E. Knuth and P. MacKay, 'Mixing Right-to-left Texts with Left-to-right Texts', *TUGboat*, **8** (1), 14–25 (1987).