בְּרֵאשִׁית בָּרָא אֱלֹהִים אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃
יִדְרֹשׁוּן, שֹׂנֵא, וְלַשֹּׂרֵקָה, מַצּוֹת, מִצְוֹת

Figure 1: A sample of the output of the vowel-placement mechanism described in this paper. The top line shows the first few words of Genesis. The bottom line shows a few words that are traditionally considered difficult to set [8]. The typeface is a version of Hadassa, designed in the 1950's by Henri Friedlaender.

בְּרֵאשִׁית בָּרָא אֱלֹהִים אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ
יִדְרֹשׁוּן, שֹׂנֵא, וְלַשֹּׂרֵקָה, מַצּוֹת, מִצְוֹת

Figure 2: Another example; the font is Omega Serif Hebrew (a version of a typeface called David, designed by Ismar David in the 1950's).

A Simple Technique for Typesetting Hebrew with Vowel Points

Sivan Toledo

1 Introduction

This paper describes a simple mechanism for typesetting Hebrew with vowel points. Hebrew uses a large set of accents that represent vowels, consonant modifiers, and cantillation instructions. These accents are placed above, below, or inside letters; a single letter can carry several of accents. The solution that we describe, which is designed for PostScript [2] output devices, leaves the placement of the accents to the output device. TEX regards the accents as zero-width characters and does not process them in any special way. Samples of the output is shown in Figures 1 and 2.

The paper only addresses the issue of typesetting vowel and consonant modifiers, known collectively in Hebrew as *nekudot*, or points. We refer to this group of accents as vowels in the rest of the paper. Cantillation marks are only used in printing Biblical texts, they are missing in most fonts, and they complicate the placement of accents. In particular, a cantillation mark attached to one letter may require ajdustments in the positioning of adjacent letters and/or marks attached to them. Vowels, on the other hand, are widely used in Hebrew printing, and are included in most of the Hebrew fonts that have been produced in recent years.[1] A vowel mark attached to a letter does not affect the positioning of other letters or the marks attached to other letters. Without vowels, it is only possible to determine the meaning and pronunciation of many words from their on the context. Although all proficient Hebrew readers can read most texts without vowels, vowels are still used in printing of liturgical texts, poetry, and texts for beginning readers. Vowels are also used in texts for proficient readers to specify the pronunciation of difficult words.

The tehchnique described in this paper is designed for typesetting Hebrew with vowel marks. The placement of a vowel mark is determined by the specific letter it is attached to and possibly by other marks attached to the same letter. The technique does not allow the positioning of marks or letter to depend on adjacent letters and their marks, so it may be insufficient for setting Hebrew with both vowels and cantillation marks.

Two factors contribute to the difficulty of typesetting Hebrew with vowels. First, the large number of possible letter-vowel combinations makes it impractical to use a separate glyph for each possible combination. Second, every vowel needs to be placed at a particular location relative to each letter. (one can decompose the set of vowels into classes of vowels with the same placement, but there are at least 7 such classes). Generally speaking, a well chosen placement visually centers the vowel with respect to the letter, but there are exceptions.

Because of these technical difficulties, one often finds texts in which the vowels are placed incorrectly. In some cases, the placement is just off center, but in other cases it is completely incorrect, to the extent that the vowel appears on the wrong side of the letter or under the wrong letter. This happens even in Bibles (see [8] for examples) and in books that were recently published by major Israeli publishers, as shown in the samples in Figure 3.

This paper does not explain how vowel points should be set; for information on this topic, see [5, 6, 8, 11, 14]. For further information on Hebrew typography in English, see [12]; this article is a bit

[1] Nearly all the TrueType fonts that come with the Apple Macintosh's Hebrew Language Kit (`www.apple.com`), with Microsoft's Hebrew Windows, and with Hebrew word processors include vowel points (Microsoft's Hebrew products are available outside Israel from Glyph Systems, `www.glyphsys.com`). So do nearly all the commercial PostScript and TrueType fonts that are produced by Elsner and Flake (`fontinform@t-online.de`; distributed in Israel by Panergy, `www.inter.net.il/~panergy`), by Studio Rosenberg in Israel (with the exception of their display typefaces; `mailto:master_f@netvision.ac.il`), and by Monotype (`www.monotype.com`).

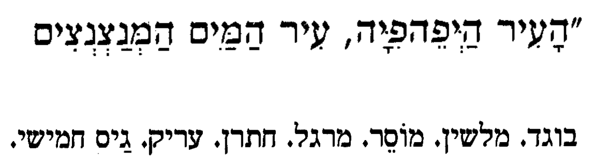

Figure 3: Examples of the poor setting of vowel point in books that were recently published by major Israeli publishers. The top sample is a true typesetting disaster, with totally incorrect placement of many vowels. The bottom sample is generally well set, except for the leftmost vowel sign which is set too much to the right.

dated, but is widely available in libraries. The collection [13] contains more up-to-date information in Hebrew.

2 An Overview of the Vowel Placement Mechanism

This section describes the mechanism that I have developed for placing Hebrew vowel.

The main idea behind the mechanism is to leave the precise placement of the vowels to the PostScript font program that the output device uses, rather than require the main typesetting engine, TeX, to place these accents. To TeX, vowels appear to be regular characters with zero widths. The advantages and disadvantages of this technique over other possible vowel-placement mechanisms are discussed in Section 4.

In the source text file, the vowels are placed after the base letter, in accordance with Unicode usage [14] (and unlike the current text-entry mechanism for Latin accents in TeX). When TeX--XeT(a bidirectional version of TeX) processes the file, the accents are placed to the left of the base letter. Once the dvi file is processed by a dvi-to-PostScript processor, they are placed in the PostScript output file *before* the base letter, since the order of the characters of a word within right-to-left text is reversed.

The PostScript font that the Hebrew is set in is responsible for vowel placement. We use a Type 3 font [2], in which the rendering of each glyph causes the invocation of an arbitrary PostScript procedure. When this procedure is asked to render a vowel mark, it does nothing, except *remember* (raise a boolean flag) that this vowel must be set when the next base letter is to be rendered. When the procedure is asked to render a base letter, it renders the base letter and then renders all the vowel points whose flags are raised, and clears the flags. Since the rendering of the base letter and vowel points is now done together in the same PostScript procedure, we can easily control the placement of the vowels with respect to the letter.

This technique is insensitive to the ordering in the input file of the vowels that apply to a single base letter, which is consistent with the Unicode specification.

The Type 3 font renders the letters and vowel points by calling another font program, not by specifying the actual shapes of the glyphs. The Type 3 font therefore behaves in much the same way that a virtual TeX font behaves, but it is considerably more flexible, since it can use all the facilities of the PostScript language. In particular, current virtual TeX files are limited to 256 glyphs, which prevents them from being used to specify all the possible letter-vowel combinations in Hebrew.

There are several advantages to calling another font to render the glyph rather than specifying their shapes in the Type 3 font. First, since we use unmodified Type 1 [1] and TrueType [9] fonts (encapsulated in a Type 42 PostScript font [3]), we preserve their hints and avoid license violations that may be associated with a conversion of the outlines to a Type 3 font that is subsequently modified. We also gain in PostScript rasterization speed, since the glyphs of the Type 3 must not be cached (the rasterizer must call the font to "render" vowel points, and the shape of base letters depends on their vowels). But the glyphs of the Type 1 or Type 42 font that the Type 3 font calls are cached. Still, the placement of vowels by the Type 3 slows down rasterization considerably compared to the rasterization of a text set in a Type 1 font, but this problem can be alleviated by optimizing the procedures of this Type 3 font, or by conversion to PDF format [4], as explained below. Early experiments with a Type 3 font that both placed vowels and specified the outlines of glyphs revealed that when PostScript files that use such fonts are distilled to PDF using Adobe Distiller, the resulting PDF file encodes the outline of every glyph that appears on the page separately. That is, in a document that used 100 different glyphs 100,000 times, the PDF file would include 100,000 glyph outlines, not just 100. This produces a large file that renders slowly. In contrast, when the Type 3 font calls a Type 1 font, the PDF file encodes only the placement of every glyph on the page, and rendering speed is essentially the same as the speed of rendering a page set in a Type 1 font.

3 Technical Details

This section describes the implementation of the Type 3 font that places vowels.

The BuildChar and BuildGlyph procedures of the font are fairly standard: the BuildChar procedure calls the BuildGlyph, and BuildGlyph calls the appropriate procedure from a CharProcs dictionary, after pushing on the stack the CharStrings dictionary itself. It also pushes a mark, so that the stack can be cleaned up later.

```
/BuildGlyph {
  mark
  3 1 roll
  exch /CharProcs get
  dup
  3 -1 roll
  2 copy known not {pop /.notdef} if
  get exec
  cleartomark
} bind def

/BuildChar {
  1 index /Encoding get exch get
  1 index /BuildGlyph get exec
} bind def
```

The CharProcs dictionary includes procedures that render base letters, that "render" vowels (in effect, these just raise a flag), and procedures that perform the actual rendering of vowels based on the state of the flags. The dictionary also includes the flags themselves.

```
/CharProcs 200 dict def
CharProcs begin

% vowel flags
/sheva_flag       false def
/hatafsegol_flag  false def
...
```

When the PostScript rendering engine needs to render a vowel, it calls a procedure similar to the one below, which simply sets the appropriate flag. The font metrics file describing this font must reflect, of course, the fact that the width of the vowels is 0. (It does not matter what the width of the vowels in the font that is used for actual rendering is.)

```
/sheva {
  % zero-width character, do not cache
  0 0 setcharwidth
  % set the flag
  /sheva_flag true put
} def
```

When the PostScript rendering engine needs to render a base letter, it calls a procedure like the next one. The procedure first renders the letter by calling another font, and then calls procedures that render vowels, if the appropriate flags are set. The horizontal and vertical offsets for each vowel are specified separately. The reason that the width of the base letter is specified as 1/1000 the width specified in the afm metrics file, and that the scaling of the font that renders the glyph is 1, is that we specify the font transformation matrix (the font's `/FontMatrix`) for this Type 3 font as [1 0 0 1 0 0].

```
/bet {
  % width taken from .afm file
  0.614 0 setcharwidth
  % render using another font
  0 0 moveto
  /MonotypeHadassah findfont
  1 scalefont setfont
  (\341) show
  % now set placement for each vowel and call
  % /draw... to render it if necessary.
  % The sheva, for example, is offset 0.250
  % units to the right with respect the a bet.
  dup 0.250 0.000 4 -1 roll /drawsheva get exec
  dup 0.300 0.000 4 -1 roll /drawhiriq get exec
  ...
  dup 0.220 0.000 4 -1 roll /drawtsere get exec
  % this vowel, holam, is moved both
  % horizontally and vertically
  dup -0.04 -0.08 4 -1 roll /drawholam get exec
} def
```

The next set of procedures includes the ones that test the flags, clear them, and render the glyphs when necessary.

```
/drawsheva {
  % arguments are already on the stack
  moveto
  % get the flag, leave on stack
  dup /sheva_flag get exch
  % clear the flag
  dup /sheva_flag false put exch
  % if set, render
  {dup /sheva_glyph get exec}
  if
} def
...

/sheva_glyph {
  /MonotypeHadassah findfont
  1 scalefont
  setfont (\300) show
} def
...
```

4 Discussion and Comparison to Related Techniques

The vowel-placement technique described in this paper allows for accurate placement of vowels of Hebrew. It was designed for use with TeX--XeT, but it can also be used with other typesetters that produce similar PostScript output. It can be used with any unmodified Type 1 or TrueType font (encapsulated in a Type 42 font). The technique requires only one software component besides TeX--XeTand a Hebrew PostScript font: a Type 3 PostScript font that places vowels. The end user simply types the Hebrew text with vowels following the base letter (as prescribed by Unicode) and runs the file through TeX--XeTand a dvi-to-PostScript processor. The resulting output file can be converted to a high-quality PDF file that render quickly. Since the technique does not use any TeX macros, it does not interfere with any, and texts with vowels can be freely used in moving arguments, indices, etc.

Producing a new Type 3 font that specifies vowel placement takes a few hours, even without interactive visual tools (I used a text editor and a PostScript previewer). The production process can probably be sped up using an interactive placement editor, or at least a table-driven script that would generate the actual font. Producing such fonts requires no special TeX expertise. The Type 3 font that places vowels for the Omega Serif Hebrew font is freely available from the author.

The technique has other potential advantages, which I have not yet exploited, but are worth mentioning. It is possible to choose narrow vowel glyphs for narrow letters, thus ensuring that the vowel does not extend beyong the letter. It is possible to render the vowels in a different color than letters; I think that this may be useful in large sizes, where rendering the vowels in gray, so as to emphasize the base letters.

The main disadvantages of this technique are that the resulting output files render somewhat slowly and that it is fragile. The PostScript rasterization speed is slow because the PostScript interpreter computes glyph placements algorithmically. As explained above, the problem can be alleviated by optimizing the Type 3 fonts or by conversion to PDF, especially for documents that are designed to be read on a computer screen. The method is fragile because it assumes that TeX and the dvi processor always place the vowels before the base letter in the PostScript file, and that the PostScript interpreter would render the glyphs in the order in which they appear in the file. These assumptions are true today, but they may change in the future. Still, I have not encountered any robustness problems yet.

Since letter-vowel combinations are rendered by a font program, they must be rendered with no information about nearby glyphs. For example, it is not possible to render vowels based on the vowels of the letters to the right and left of the current one, or based on whether the letter appears at the beginning or end of a line. This is not a serious limitation for typesetting Hebrew with vowel marks, but it a limitation when typesetting biblical texts with both vowel and cantillation marks.

There are other solutions and partial solutions to the vowel-placement problem. One common partial solution is to use fonts that have some precombined letter-vowel glyphs. A set of around 50 or 60 precombined glyphs, which are treated by the typesetting software as ligatures, ensure an accurate placement of the most difficult combinations. The other combinations are produced by overstriking a base letter or one of the precombined glyphs with vowel glyphs. Overstriking usually places the vowel slightly off center (because a single overstriking vowel glyph is used for all base letters). The typographical quality of the result is not high, but it is not a disaster either. Also, to ensure that ligatures are used, the user must type vowels in a specific order, which is inconsistent with Unicode usage. For example, if the font includes a ligature for the letter yod with a dagesh mark, the user must type the dagesh before other vowels that apply to the yod. This type of solution appears to be fairly common in commercial publishing in Israel.

Another solution, specifically for TeX, was developed by Haralambous [5, 6]. His system, which sets both vowel and cantillation marks, uses a preprocessor and a special font to place these accents. I do not know whether the system can be adapted to other fonts. While Haralambous's system is more robust and flexible than the one I describe, my approach is simpler to implement and use.

Haralambous is also one of the developers of Omega [7] (with Plaice), a system for multilingual typesetting that is based on TeX, uses a 16-bit character set (Unicode), adds to TeX another processing mechanism that is separate from macros, and has more flexible virtual fonts. Omega should enable typesetting Hebrew with vowel points.

The introduction of OpenType [10] fonts will probably encourage the development of additional solutions. OpenType fonts, which were developed by Adobe and Microsoft, are essentially enhanced TrueType fonts that can contain Type 1 outlines and advanced typographical layout information, similar in spirit to the information that Apple's TrueTypeGX fonts can contain. OpenType fonts are already used in Arabic and Far Eastern versions of Microsoft Windows, and they are supposed to be used in future versions of all versions of Windows, including Hebrew. It is possible that Hebrew OpenType fonts with enough information for accurate placement of vowel points will be produced. There are currently no such fonts, however, and no programs that can use them. Converting OpenType fonts and the typographic information that they contain to a format that TEX can use is certainly a complex task.

All in all, I found this technique to be a remarkably simple solution to a complex problem.

5 Acknowledgements

Thanks to Barbara Beeton, Yannis Haralambous, and an anonymous referee for helpful comments.

References

[1] Adobe Systems Incorporated. *Adobe Type 1 Font Format.* Addison Wesley, 1990.

[2] Adobe Systems Incorporated. *PostScript Language Reference Manual, 2nd Edition.* Addison Wesley, 1990.

[3] Adobe Systems Incorporated. *The Type 42 Font Format Specification.* Technical Note 5012, 1993.

[4] Tim Bienz, Richard Cohn, and James R. Meehan *Portable Document Format Reference Manual, Version 1.2.* Adobe Systems Incorporated, 1996.

[5] Yannis Haralambous. Typesetting the Holy Bible in Hebrew, with TEX. *TUGboat*, 15(3): 174–191, 1994. Also appeared in the *Proceedings of EuroTEX 1994*, Gdańsk, 1994.

[6] Yannis Haralambous. "Tiqwah": a typesetting system for biblical Hebrew, based on TEX. *Bible et Informatique*, 4:445–470, 1995.

[7] Yannis Haralambous and John Plaice. Omega, a TEX extension including Unicode and featuring lex-like filtering processes. *Proceedings of EuroTEX 1994*, Gdańsk, 1994.

[8] Eliyahu Koren. [The letter as an element in the design of sacred books]. In Hebrew. In [13], pages 85–90. אליהו קורן. האות כיסוד בעיצוב ספרי קודש.

[9] Microsoft Corporation. *TrueType Specification, Version 1.66.* Available online at `www.microsoft.com/typography`.

[10] Microsoft Corporation. *OpenType Specification, Version 1.1.* Available online at `www.microsoft.com/typography`.

[11] Gershon Silberberg with contributions by Moshe Spitzer, Meir Ben-Yehuda, Shmuel Perez, and Arie Lotan. Principles of Printing. Irgun Mifale ha-Defus be-Yisrael, Tel Aviv, 1968. גרשון סילברברג, בהשתתפות משה שפיצר, מאיר בן־יהודה, שמואל פרץ, ואריה לוטן. **תורת הדפוס**. ארגון מפעלי הדפוס בישראל, תל־אביב, תשכ״ב.

[12] Moshe Spitzer. Typgraphy. *Encyclopedia Judaica*, 15:1480–1488, Keter, Jerusalem, 1971. In English.

[13] Moshe Spitzer, editor. *[A Letter is Forever: A Collection of Papers on the Design of the Hebrew Letter].*In Hebrew. Second edition, Israel Ministry of Education and Culture, 1989 or 1990. משה שפיצר, עורך. **אות היא לעולם: קובץ מאמרים מוקדש לעיצוב האות העברית.** משרד החינוך והתר־בות, האגף לחינוך תורני, ירושלים, תש״ן.

[14] The Unicode Consortium. *The Unicode Standard Version 2.0.* Published by Addison-Wesley, 1996. Parts of the standard are available online at `www.unicode.org`.

◇ Sivan Toledo
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
USA
(On leave from Tel Aviv University)
`toledo@parc.xerox.com`,
`sivan@math.tau.ac.il`

 preliminary draft, September 2, 1998 9:52