A PostScript to Plain Text Converter

Todd Reed Craig Neville-Manning

June 5, 1996

Introduction

This article describes the implementation of *PreScript*, a PostScript to plain text conversion utility. Many similar utilities already exist, most notably the `ps2ascii.ps` program distributed with the freely available *Ghostscript* PostScript interpreter. *PreScript* is relatively robust compared to other converters and offers a number of features not found in any other utility, including the detection of paragraph boundaries and hyphenation removal.

PostScript Text Rendering

Converting *from* PostScript is quite different than converting from other traditional document formats, because PostScript is not a *data* format, but a full-fledged programming language. PostScript documents are really PostScript programs that are executed by an interpreter to produce some result. Herein lies the difficulty of converting PostScript to ASCII text – the content of a PostScript program is not the content of the document, but rather instructions on *how* to render the document. As a contrived example, consider the PostScript sample and output in Figure 1. This program displays the 6th number of the Fibonacci sequence but the string "8" appears nowhere in the program. Clearly, any syntactic approach to PostScript conversion will be unreliable.

The fact that PostScript is a programming language, however, allows for an alternative approach. By redefining some standard routines prior to executing a PostScript program, we can change the normal behaviour of that program. For rendering text, PostScript provides a group of commands, collectively called the *show* commands: `show`, `kshow`, `widthshow`, `ashow`, and `awidthshow`. Appropriate redefinitions of these commands that print their string argument to a file, along with the *current point*[1], provide all the information to construct an ASCII version of the PostScript document.

Appendix A shows the PostScript prologue `prescript.ps` that implements `psxShow`, which is called each time a *show* command is executed. For

[1] The PostScript interpreter maintains a current point, the endpoint of the most recently rendered object.

```
/Buffer 10 string def

/fib {
dup
1 eq { }
  { dup 2 eq
    { pop 1 }
    { dup 1 sub fib exch 2 sub fib add }
  ifelse }
ifelse } def

/Times-Roman findfont 50 scalefont setfont
10 10 moveto
(Fib[6] = ) show 6 fib Buffer cvs show
```

$\boxed{\text{Fib[6] = 8}}$

Figure 1: A contrived PostScript example.

each execution, `psxShow` writes to a data file the current point, the string being rendered, and the current point after the string is rendered. To update the current point, the appropriate "original" *show* function is called. The data file is formatted so it can be easily parsed by a post-processing program; each call to `psxShow` is recorded as a tab delimited line of the form:

`S` x_0 y_0 *string* x_1 y_1

where (x_0, y_0) is the current point prior to rendering *string*, and (x_1, y_1) is the current point after. `Prescript.ps` also redefines `showpage` to write a `P` to the data file, indicating a page boundary. Figure 2 shows the generated data file for this sentence.

The `prescript.ps` prologue is similar to (and based on) the `ps2ascii.ps` program that is distributed with *Ghostscript*. While `ps2ascii.ps` attempts to track font changes, this extra baggage effects robustness. The KISS adage applies here. During initial experiments, `ps2ascii.ps` failed on 9 of 13 test files. No claim to perfection is made, however. While `prescript.ps` will fail on some PostScript files, it handily beats `ps2ascii.ps` on robustness.

Text Statistics

Let $f_1, \ldots, f_n$ be the ordered sequence of text fragments generated by the PostScript program. It is assumed that the PostScript program renders text in reading order within each page, and that $x_0 <= x_1$ for each fragment (*i.e* x val-

```
S       2074    4011    Figur   2273    4011
S       2272    4011    e       2312    4011
S       2336    4011    ??      2410    4011
S       2433    4011    shows   2665    4011
S       2691    4011    the     2806    4011
S       2830    4011    generated       3197    4011
S       3220    4011    data    3382    4011
S       515     4111    %aele  631     4111
S       652     4111    for     758     4111
S       778     4111    this    912     4111
S       934     4111    sentence.       1270    4111
```

Figure 2: Data file produced by redefinitions of the *show* commands.

ues increase to the right)[2]. We define the samples ΔX, ΔY, and C as follows:

$$\begin{aligned}
\Delta X &= \{x \mid x = f_{i+1}.x_0 - f_i.x_1,\ x < 0,\ i = 1, \ldots, n-1\} \\
\Delta Y &= \{y \mid y = f_{i+1}.y_0 - f_i.y_1,\ y \neq 0,\ i = 1, \ldots, n-1\} \\
C &= \left\{c \;\middle|\; c = \frac{f_i.x_1 - f_i.x_0}{\text{length}(A.string)},\ i = 1, \ldots, n\right\}
\end{aligned}$$

ΔX is the set of negative horizontal movements of the current point between fragments. ΔY is the set of nonzero vertical movements of the current point between fragments. C is the set of average character widths for each fragment[3]. With the additional assumption that the document contains predominantly regular text, ΔX and ΔY can be used to guesstimate the global line width and line spacing respectively.

Let $\phi^n(X)$ be the n-th most frequent value in X ($\phi^1(X)$ is the mode of X). We then compute global values for character width (c), line width (w), and line spacing (s) as follows:

$$c = \phi^1(C) \tag{1}$$
$$w = |\phi^n(\Delta X)| \geq 5c \tag{2}$$
$$s = \phi^m(\Delta Y) \geq c \tag{3}$$

In equation 2 we choose the smallest integer n such that $w \geq 5c$. Usually, the most frequent ΔX, $\phi^1(\Delta X)$, is the line width, but if the text contains many mathematical equations where symbols don't have a strict left-to-right reading,

[2]We have processed over 14000 PostScript files and have not found an example for which these assumptions are invalid.

[3]Because ligatures appear as a single character, the computation of average character width may be larger than the actual average character width. The relative infrequency of ligatures, however, makes the error insignificant.

we can expect numerous small ΔX values that should be ignored. For similar reasons, in equation 3 we choose the smallest integer m such that $s \geq c$.

If the document contains copious amounts of irregular text, such as mathematical equations or program listings, this approach to determining the global line width and line spacing may fail. Indeed, there not be a *global* line width or line spacing. A program listing, for example, has no apparent line width; a technical specification may have two apparent line spacings: one for the main text and another for annotations.

Conversion

Having computed c, w and s, the following rules are applied to convert the list of fragments $f_1, \ldots, f_n$ to a reasonable plain text representation for the document. In the following descriptions, assume that A and B are consecutive fragments.

End of line B starts a new line if

$$A.x_1 - B.x_0 > 2c \text{ and } A.y_1 \neq B.y_0$$

or

$$|B.y_0 - A.y_1| > 2c$$

Word Spacing A space is inserted between A and B if $B.x_0 - A.x_1 > 0.3c'$, where c' is a locally computed average character width:

$$c' = \min\left\{\frac{A.x_1 - A.x_0}{\mathrm{length}(A.\textit{string})}, \frac{B.x_1 - B.x_0}{\mathrm{length}(B.\textit{string})}\right\}$$

If no space is to be added between A and B, then B is concatenated to A: $A.\textit{string} \leftarrow A.\textit{string} + B.\textit{string}$, $A.x_1 \leftarrow B.x_1$, $A.y_1 \leftarrow B.y_1$. This heuristic was stolen from DEC's *pstotext* program [1].

Paragraph boundaries Paragraph boundaries are detected when the inter-paragraph spacing is greater than the line spacing. If the spacing between two lines exceeds $1.1s$, a new paragraph is started. This heuristic fails when the document has more than one commonly used line spacing scheme.

Even when the line spacing doesn't indicate a new paragraph, under certain strict conditions, a paragraph boundary may be identified by indentation. Suppose we have three text lines, L_0, L_1, and L_2 (L_2 being the bottom-most line). A new paragraph starts at L_1 if all of the following conditions are satisfied:

- L_1 is shifted right relative to L_0 and L_2:

$$L_1.x_0 > L_0.x_0 + 2c$$

and

$$L_1.x_0 > L_2.x_0 + 2c$$

- L_1 starts with a capital letter.
- L_2 is the longest line of L_0, L_1, and L_2.
- L_1 is almost as long as L_2:

$$L_1 \geq 0.9L_2$$

Explicit line breaks An explicit line break is inserted after lines whose length is less than $0.8w$. If something is known about the document format, additional line breaks may also be inserted. For example, entries in the table of contents are often easily identified by repeated periods.

Page reversal It is often possible to detect page numbering by looking for isolated numbers at the top or bottom of each page. If page numbers are found for a significant portion of pages and the page numbers are decreasing, then the pages can be reversed[4]. Page reversal is not currently implemented in *PreScript*.

Odds and Ends

PreScript implements two additional features: ligature and hyphenation removal. Ligatures are "compound characters" – two or more characters joined into one character. In PostScript, ligatures are usually encoded as single characters in a special font. In Figure 2, for example, the word "file" appears as three characters: `%ae`[5], `l`, and `e`; the single character `%ae` represents "fi". Implementing this translation is trivial but dependent on the source document. Ligature translations valid for documents created with TEX are not the same as those valid for Word documents. Worse, some ligatures appear as common characters making their detection impossible without tracking font changes[6].

Another easy feature to implement is hyphenation removal. If the last word of a line ends with a "-" character, the hyphen is removed and the first word of the following line is concatenated. Of other converters, only *ps2html* [2] implements hyphenation removal; it even uses an online dictionary to ensure the new word is valid. One shortcoming of the hyphenation removal in *PreScript* is that it does not work across page boundaries because the first and last lines of a page are often, but not always, headers and footers.

[4] The obvious solution to detecting reversed pages – examining DSC comments (%%PageOrder or %%Page) in the PostScript source – produces completely unreliable results.

[5] The notation %*xx* is used to represent the character with hexadecimal code *xx* – necessary to represent tabs and newlines, and non-ASCII characters.

[6] Some utilities do track font changes, but the added complexity of doing so makes these programs considerably less reliable. The exception is DEC's *pstotext* [1], which is about as robust as *PreScript*, but up to 10 times slower.

Problems

- Indented text comes out non-indented. This is disastrous for program listings.
- No de-hyphenation across page boundaries.

References

[1] Andrew Birrel and Paul McJones. The pstotext program. http://www.research.digital.com/SRC/virtualpaper/pstotext.html. Digital Equipment Corporation.

[2] Jerry Whelan. ftp://bradley.bradley.edu/pub/guru/ps2html/ps2html-v2.html.

A The `prescript.ps` Program

```
%    prescript.ps 0.1 -- A PostScript to text converter
%    Copyright (C) 1996  Craig Neville-Manning, Todd Reed
%
%    This program is free software; you can redistribute it and/or modify
%    it under the terms of the GNU General Public License as published by
%    the Free Software Foundation; either version 2 of the License, or
%    (at your option) any later version.
%
%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%    GNU General Public License for more details.
%
%    You should have received a copy of the GNU General Public License
%    along with this program; if not, write to the Free Software
%    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
%
%
%
% This program spits out data about the text contained in a Postscript
% file.
%
% Output format:
%
% S x0 y0 string x1 y1
%
%      (x0, y0) is the position of the current point before string is
%      printed, and (x1, y1) is the position after.  These numbers are
%      integers.  string is the text to be rendered.  Fields are separated by
%      tab characters.  The character % is used as a control character to
%      represent tabs, newlines and the % character itself inside string.
%      The format is %xx, where xx is a hex number for the ASCII code.
%
%      For unexplainable reasons, recursive calls to show may cause this line
```

```
%      to be embedded in another.
%
% P
%
%      Indicates a showpage was executed.
%
%
% Run this program as
%
% gs -q -dNODISPLAY -soutfile=<output> prescript.ps <psfile> quit.ps
%
30 dict begin

/psxFile outfile (w) file def
/psxCharBuffer 1 string def
/psxBuffer 1024 string def
/psxPrint { psxFile exch writestring } bind def
/psxPrintNumber { round cvi psxBuffer cvs psxFile exch writestring } bind def

% This routine is for an experimental feature to extract embedded EPS
% files in the Postscript file, render them a GIFS, and later embed them
% in, for example, an HTML file.  The psxEPS command is introduced into a
% PostScript file by xeps.
%
/psxEPS { (E\t) psxPrint psxPrintNumber (\n) psxPrint } def

/psxEncodedPrint {
  {
    dup 9 eq
    1 index 10 eq or
    1 index 37 eq or
      { psxFile 37 write psxCharBuffer 0 3 -1 roll
 put psxFile psxCharBuffer writehexstring }
      { psxFile exch write  } ifelse
  } forall
} bind def

/psxShow {
(S\t) psxPrint

currentpoint exch
psxPrintNumber (\t) psxPrint psxPrintNumber

(\t) psxPrint
1 index psxEncodedPrint
(\t) psxPrint
systemdict exch get exec

currentpoint exch
psxPrintNumber (\t) psxPrint psxPrintNumber

(\n) psxPrint
} def

/show { /show psxShow } def
/kshow { /kshow psxShow } def
/widthshow { /widthshow psxShow } def
```

```
/ashow { /ashow psxShow } def
/awidthshow { /awidthshow psxShow } def

/setpagedevice { pop } def
/setcolor { pop } def
/settransfer { pop } def
/setgray { pop } def
/setgraypattern { pop } def

/showpage { psxFile (P\n) writestring } def
```