



Peer Reviewed

Title:

Born Broken: Fonts and Information Loss in Legacy Digital Documents

Author:

[Brown, Geoffrey](#), Indiana University
[Woods, Kam](#), Indiana University

Publication Date:

10-05-2009

Series:

[iPRES 2009: the Sixth International Conference on Preservation of Digital Objects](#)

Publication Info:

iPRES 2009: the Sixth International Conference on Preservation of Digital Objects, California Digital Library, UC Office of the President

Permalink:

<http://www.escholarship.org/uc/item/53z897zb>

Multimedia URL:

<http://www.cdlib.org/services/uc3/iPres/video.html?file=ipres/Brown&title=Geoffrey%20Brown%3A%20Born%20Broken%3A%20Fonts%20and%20Information%20Loss%20in%20Legacy%20Digital%20Documents>

Abstract:

For millions of legacy documents, correct rendering depends upon resources such as fonts that are not generally embedded within the document structure. Yet there is significant risk of information loss due to missing or incorrectly substituted fonts. In this paper we use a collection of 230,000 Word documents to assess the difficulty of matching font requirements with a database of fonts. We describe the identifying information contained in common font formats, font requirements stored in Word documents, the API provided by Windows to support font requests by applications, the documented substitution algorithms used by Windows when requested fonts are not available, and the ways in which support software might be used to control font substitution in a preservation environment.

Supporting material:

Presentation

Copyright Information:



iPRES 2009

THE SIXTH INTERNATIONAL CONFERENCE ON THE PRESERVATION OF DIGITAL OBJECTS

Proceedings

October 5-6, 2009
Mission Bay Conference Center
San Francisco, California



California Digital Library

Born Broken: Fonts and Information Loss in Legacy Digital Documents

Geoffrey Brown and Kam Woods

Department of Computer Science, Indiana University
150 S. Woodlawn Ave.
Bloomington, IN 47405-7104

Abstract

For millions of legacy documents, correct rendering depends upon resources such as fonts that are not generally embedded within the document structure. Yet there is significant risk of information loss due to missing or incorrectly substituted fonts.

In this paper we use a collection of 230,000 Word documents to assess the difficulty of matching font requirements with a database of fonts. We describe the identifying information contained in common font formats, font requirements stored in Word documents, the API provided by Windows to support font requests by applications, the documented substitution algorithms used by Windows when requested fonts are not available, and the ways in which support software might be used to control font substitution in a preservation environment.

Overview

Q. I need ‘font x’. ... Can you tell me where I can get it?
A. This is very unlikely, as there are over 100,000 digital fonts in existence.¹

Doubtless, many readers have witnessed PowerPoint presentations where the slides were clearly missing glyphs (visible characters) or were otherwise poorly rendered. In most cases, this unhappy event is the direct result of copying the presentation from the machine upon which it was created to a machine provided for the presentation without ensuring that the target machine has the required fonts. The reason is not always clear to the presenter because Microsoft Office performs font substitution without warning.

Annoying font substitutions occur frequently. For example, symbols such as apostrophes and quotations are rendered with the “WP Typographic Symbol” font in WordPerfect Office 11. When these documents are migrated to Microsoft Word, this font dependence is preserved, and when the documents are rendered on machines without this font, these symbols become “A” and “@” as in: “in the AStrategies and Assessment@ Column”.²

The degree of information loss due to substitution depends both upon the importance of the glyphs substituted

and their frequency. For example, corporate logos are often implemented with dedicated fonts containing a single glyph. There may be no substantive information loss from a missing logo for most purposes. In contrast, substitution for mathematical symbols may result in total information loss. For example, our experimental data include 9 documents with program listings for the Texas Instruments TI-83 series calculators rendered with the Ti83Pluspc font which provides various mathematical symbols; these program listings are incomprehensible when rendered without this obscure font. This is illustrated in Figure 1. Compounding the problem, Texas Instruments has published a variety of calculator fonts with different internal names and possibly incompatible glyphs.

```
Lbl A:ClrHome
If N<4:Goto 0
CubicReg L1,L2:3→T:"aX3+bX2+cX+d"→Y1
Disp "aX3+bX2+cX+d"
```

Correctly Rendered

```
Lbl A:ClrHome
If N<4:Goto 0
CubicReg L ,L,:3ūT:"aX3+bX2+cX+d"ūY
Disp "aX3+bX2+cX+d"
```

Default Substitution

Figure 1: T183plus font samples

In the process of preparing this paper we found several documents with barcodes rendered using the font “Barcode 3 of 9 by request”. When rendered with font substitutions, the bar codes of the numbers are replaced by Arabic numerals. Thus, the substitution preserved the numeric meaning, but not the functional ability to be scanned! Unfortunately, there are many barcode fonts and we were unable to find the required font; however, we were able to find a suitable substitute and to configure Office to accept our substitute as illustrated in Figure 2.

0260931

Default Substitution



Forced Substitution of “Code39Azalea”

Figure 2: Barcode Rendering

¹<http://www.microsoft.com/typography/FontSearchFAQ.msp>

²<http://www.wpuniverse.com/vb/showthread.php?threadid=16756>

It is difficult to create portable documents that will not suffer from font substitution when moved to another machine. Fonts are installed on a platform by the operating system, by applications, and by individuals. A user has no way to distinguish between standard fonts offered in an Office menu and those that have been installed by other applications. For example, ESRI provides fonts with its various GIS applications. While these are relatively specialized applications that are unlikely to be present on most platforms, there is nothing preventing an Office user from utilizing the associated fonts. Furthermore, even the set of standard fonts changes over time – the fonts installed by Windows XP are not identical to those installed by Vista. Because of significant differences in the machine environments, there is a high probability that transferring a document between machines will result in missing fonts.

A solution to the problem of missing fonts for document preservation requires three components – identification of missing fonts, acquisition of the fonts or suitable substitutes, and configuring a suitable rendering environment including all fonts or their substitutes. In this paper we focus on the problem of font identification. The issues we discuss include extraction of font identification data from digital documents, the use of that information to match against a database of known font identifiers, and techniques for controlling font substitution.

The central analysis in this paper is based upon two large collections of Word documents – one described in (Reichherzer and Brown 2006) gathered using glossary queries to Google, and the second exclusively gathered from “.gov” sites. The second collection was created to test an hypothesis that government documents might use more restricted font sets; however, that did not prove to be the case.

To test our ability to match font requirements with font names we gathered font name information from several major vendors and application software. Names were extracted from fonts, from published lists of fonts, and from tables of names provided by vendors.

While the results initially appear depressing – a common desktop environment can correctly render around 75% of a document collection – there are indications that with modest work the fonts required to faithfully render 92% of a collection can be readily identified. The problem of identification is unsolvable in an absolute sense – in any collection of documents there are likely to be required fonts that cannot be reasonably identified. There are simply too many fonts which have been in use and the data on font names available from font foundries or the fonts themselves too sparse to guarantee identification. Furthermore, some documents may include fonts with corrupt name information – we have seen examples of documents with indecipherable font names.

The remainder of this paper is organized as follows. We begin with a discussion of font formats (e.g. TrueType) and the available font identification information. By

gathering information from fonts and font vendors, we have created a database containing several thousand popular fonts. We then describe the information available in Office documents, and our use of open source libraries to extract the names of fonts referenced by these documents. Finally, we describe experiments to match font names extracted from Microsoft Office documents with names in our database.

Background

Fonts

A font consists of a set of glyphs indexed by codepoints (integers) within one or more codepages (a defined codepoint to a character mapping such as the latin alphabet) along with various geometric rendering information. Two fonts may be suitable substitutes if they contain similar glyphs for all codepoints. While this is frequently the case where the glyphs represent characters from common alphabets, there are many special case of symbolic characters (e.g. mathematical, scientific, or icons) where substitution of glyphs from another font destroys the underlying meaning of the document.

While there have been many font formats in use, the two most common formats for Windows platforms are PostScript and TrueType. There are still bitmapped fonts distributed with Windows for MS-DOS compatibility (FON files) which do appear to be used in some Office files. However, other than noting that they contain name strings that can be extracted, we do not discuss them further.

Although PostScript fonts appear to be in the decline, they were the dominant font format for at least a decade. There are several PostScript font formats (e.g. types 0, 1, and 2) but many of the differences relate to how glyphs are defined. For font identification, the key information provided in every PostScript font includes ASCII strings identifying the font version, family name, font name, and full name (Adobe Developer Connection 2009). The full name shows the complete name of a typeface including style and character set information, and is typically used in font menus. The font name generally contains much of the same information as the full name, but in a compressed form limited to 29 characters. There are conventions for this compression (for example a rule that reduces the “words” to a string with 5, 3 and 3 characters); however, it can be challenging to relate the font name strings with published lists of fonts (Microsoft 2009a).

TrueType was developed by Apple as a competitor to PostScript fonts and was subsequently adopted by Windows. Today, TrueType is the most common font format for Mac OS, the XWindows platform, and Microsoft Windows. The file format for TrueType is now covered by the OpenType specification (which can serve as a container for PostScript fonts). OpenType files are

organized as a set of tables. The most important tables for our work are the naming tables (name) and the table (OS/2) containing Windows metrics including the Unicode and Windows code page ranges. The name table includes various strings keyed by platform, encoding, language, and name. Platforms include Windows and Macintosh. Encodings are platform specific – on the Macintosh these correspond to script manager codes (e.g. Roman, Japanese, etc.). On Windows, common encodings include Unicode UCS-2 and UCS-4.

OpenType language IDs are platform-specific, and are used to indicate various language specific translations of the name strings. The types of names include copyright, font family, font subfamily, full font name, and PostScript names. The language used in font name strings within Office documents corresponds (where available) to the language for the Windows platform upon which the document was created; for example, “Arial Bold”, “Arial Negrita”, “Arial Vet”, and “Arial Gras” are the English, Spanish, Dutch, and French names for the same font. In gathering font data for this paper we found that only the most widely used fonts tend to have name strings in multiple languages and several large foundries provided only English names strings with the majority of their fonts.

Fonts in Windows and Word

In this paper we concentrate on the legacy Microsoft Word binary format. While we have performed no specific work with other formats (e.g. WordPerfect) there is good reason to believe that the conclusions will be similar because application programs such as Word and WordPerfect depend upon the underlying operating system API for access to and rendering of installed fonts. The font information embedded in a document is ultimately based upon the information available from the system API. Thus, we begin with a brief examination of the Win32 font functionality as described in the MSDN documentation (Microsoft Developer Network 2009). A complete analysis examining the APIs of other operating systems (e.g. for the Macintosh) is beyond the scope of this paper.

The central data structure used by an application to exchange font information with the Windows operating system is the “logical font” or LOGFONT structure that is used to describe the most significant features of a font. Applications create LOGFONT structures to request that Windows find a matching font and Windows enumerates available fonts for applications by generating LOGFONT structures. The structure provides information such as weight, orientation, and style (e.g. script, decorative, roman) as well as a (maximum) 32 character (Unicode or ASCII) name for the font. The metrics provide geometric information such as pitch and width, style information such as italic or underlined, and information about the range and type of the character set supported (e.g. ANSI, Symbol, Turkish). The type information can be used to distinguish

“Raster fonts”, “Vector fonts”, “True-Type fonts”, and “Downloadable fonts”.

Word documents store their knowledge about fonts in a similar structure table of “font family names” (FFN), which include the name string (from the LOGFONT structure), a flag indicating whether the font is TrueType, the character set, the font weight and style, PANOSE number, and a 20-byte “font signature” (Microsoft 2009c). The name string is either in UTF-16 or ASCII depending upon the version of Word creating a document. The font signature indicates the Unicode (16 bytes) and Windows code pages (8 bytes) for which the font contains glyphs. Similar data exist in the OS/2 metrics table for OpenType fonts and might be useful for verifying that a font file matches the font referenced by a Word document.

PANOSE numbers were developed in the 1980’s (Bauermeister 1987) as a mechanism for classifying fonts with the explicit goal of identifying good substitutes. While TrueType fonts include PANOSE numbers, and Word retains those numbers in its font tables, published research suggests that PANOSE has not been widely implemented correctly – many fonts have “default” values (Impson 2005). Furthermore, where PANOSE is correctly used the information is likely to be of little added value for font identification because these fonts tend to be those distributed by major vendors such as Microsoft for which accurate font name data has been recorded.

In our work, we rely upon the font name strings extracted from Office documents. While every Word document contains a single font name table, not all fonts listed in this table are used by the document. Furthermore, our experience suggests that as a document evolves, Word fails to properly delete unused entries and empty or “garbage collect” name strings. Thus, extracting the set of valid name strings requires walking the document character by character.

Font Matching Experiments

In this Section we use two data sets totaling approximately 230,000 Word documents to evaluate the difficulty of identifying referenced fonts given a database of font information. The primary identification technique we utilize is based upon name matching; however, we also evaluate the utility of the other font metrics recorded in Word documents.

We had hoped to build a database of font information extracted from the fonts published by the major foundries; however, we found most foundries reluctant to provide the requested information and the cost of purchasing fonts not justified by this exploratory research.³ We resorted to combining information from a variety of sources including font files and font name information provided by several

³ We are grateful to Bitstream for providing a large collection of fonts, FontFont and URW for providing tables of font name information.

Foundry		
Source	Data Type	Number of Fonts
Adobe	Published Table	2,374
Bitstream	TrueType Fonts	1,556
FontFont	Foundry Supplied Table	11,973
URW	Foundry Supplied Table	2,358

Operating System		
Microsoft Windows + Office	Font Files	444
Mac OS X + Office	Font Files	332

Application		
Adobe PostScript 3 Fonts	Published List	103
Microsoft Applications	Published List	537
WordPerfect	TrueType Fonts	1,080

Table 1: Font Data

collections include fonts provided with various Mac OS X and MS Windows distributions, fonts distributed with applications such as Microsoft Office and WordPerfect, and fonts donated by Bitstream. The font data we collected are summarized in Table 1.

To determine the fonts used in Word documents, we wrote a custom application based upon the open source library `libwv` which is the basis for file import in Abiword and other applications (Lachowicz 2009). Through comparing key aspects of `libwv` with the published specification, we believe `libwv` is relatively correct – there are aspects of the specification that are far from clear, and specific notes point to items requiring further work. The `libwv` software provides a basic document processing function with application-specific callback functions. For our work, it was necessary only to provide a function to track the fonts specified at the beginning of a text run and a function to process each character in a text run. For each font in the document name table, we record both the codepoints used and the number of characters referencing the font. Selection of the correct font for a given character is actually done incorrectly in `libwv` – the code does not implement the “font calculation” described in Appendix B of the Word 2007 Binary File Format document (Microsoft 2009c). We corrected this error in our work. Our code generates reports including the active font names with the number of codepoints and characters from each font as well as the other font metrics recorded in the Word document.

Given the font information extracted from a collection of Word documents and a database of font names we can determine which extracted names exactly match names in the database. While there appear to be opportunities to apply various heuristics for inexact matching (for example longest matching prefixes), our examination of the names suggests there are many special

cases, thus we studied exact matching as a baseline. We matched names against three distinct name sets – our complete collection, the names extracted from an installation of Windows XP and Office 2007, and the single name “Times New Roman” which is the most common font referenced in our data sets.

Our metric for each font collection is the percentage of documents whose font requirements can be completely met (satisfied) by that collection. We compare these results with the percentage of “satisfied documents” given font collections of the N most referenced fonts for all values of N. The results for our glossary based collection (3910 fonts) are illustrated in Figure 3 and those for the “.gov” (1920 fonts) documents are illustrated in Figure 4. Although the total number of referenced fonts differs, the overall results are quite similar – roughly 31-39% satisfied by Times New Roman, 72-79% satisfied by XP and Office, and 90-94% satisfied by our more comprehensive collection. Notice that in both cases the top 100 fonts satisfy approximately 92% of the documents.

As mentioned above, exact name matching is probably too pessimistic. For example, we noticed many variations on fonts including the word “Times”. Some of this variation is due to similar fonts published by different vendors, some is due to changes in name conventions from the early bitmapped fonts to PostScript fonts to

Truetype, and some is due to significant differences. The top 10 “Times” fonts and their fraction of reference from the two document collections is illustrated in Table 2. The complete list comprises 375 fonts and 49% of all font references. Note that the values in Table 2 are calculated from the total number of references rather than the percentage of documents satisfied.

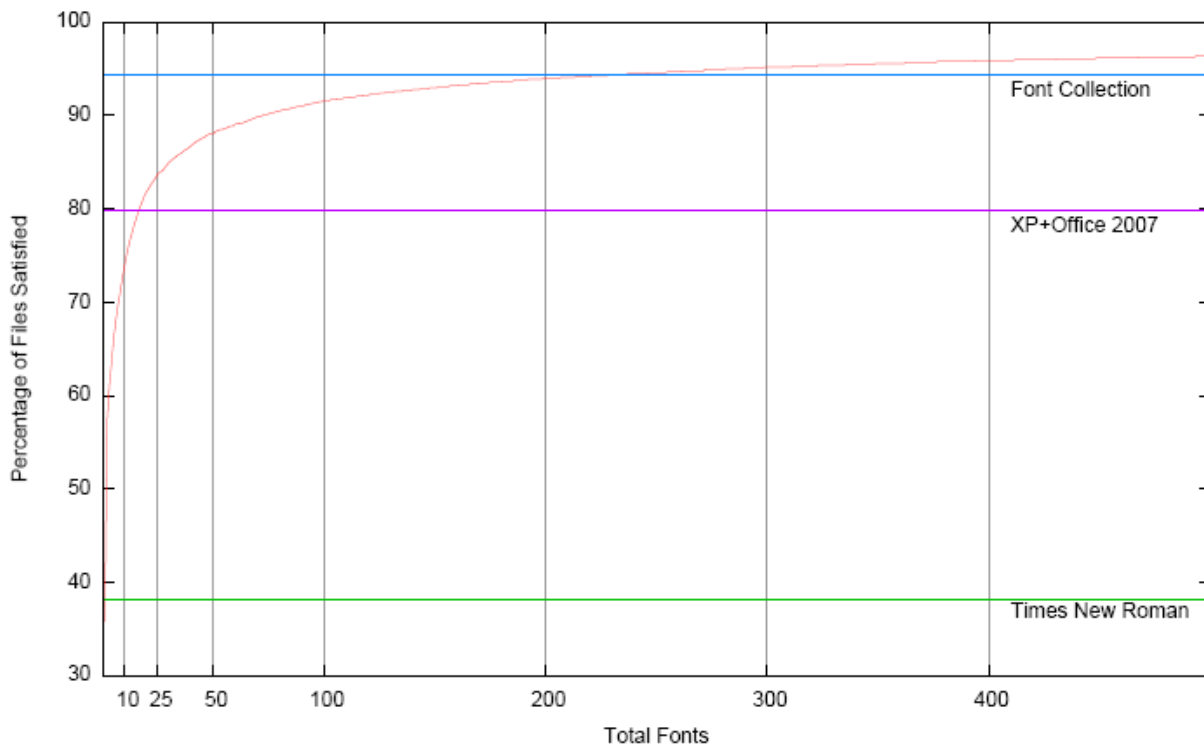


Figure 3: Font Usage for Glossary Documents

Even if all variations on Times, after suitable analysis, proved to be equivalent, the overall problem isn't significantly simplified. The extremely long tail on font usage means achieving a 95% satisfaction level for a document collection is tractable, achieving 99% may not be feasible. Furthermore, our experience suggests that finding many of the identified fonts will be quite challenging.

Ultimately, preservation of documents will require selection of suitable font substitutes either because a particular font cannot be obtained or identified. Thus, we examined other data that might aid in characterizing suitable substitutes or in determining whether a required

Times New Roman	42%
Times	3%
CG Times	1%
Times New Roman	0.5%
Times New Roman Bold	< 0.5%
Times New Roman, Bold	< 0.5%
Times New (W1)	< 0.5%
CG Times (W1)	< 0.5%
Times New Roman PSMT	< 0.5%
Times-Roman	< 0.5%
...	...
Total (375)	49%

Table 2: Times Variations. Percentages calculated from number of times each font variation is encountered.

font is critical to preserving the information content of a document.

The primary additional font metrics available from Word documents include the font family (Roman, Swiss, etc.) and pitch, the font weight, Panose number, and Code sets (Unicode and Windows). The two font families that are likely to be safely substituted are Roman and Swiss (serif and san serif fonts respectively). It is less clear whether Modern or Script can be substituted, and Decorative generally cannot be substituted. As illustrated in Table 3 and Table 4, either the font family or pitch information has a "default" value for nearly 40% of all referenced fonts. The font family categories are described in (Microsoft 2009b).

Default	40%
Fixed Pitch	3%
Variable Pitch	57%

Table 3: Font Pitch

Unicode range and Codepage information can be used to analyze font data at a finer granularity, providing a clearer picture of why a particular font may be used in a given document (e.g. in order to satisfy the need for particular glyphs) and potential guidance on selection of a suitable substitute. We intend to explore the use of such information in future research.

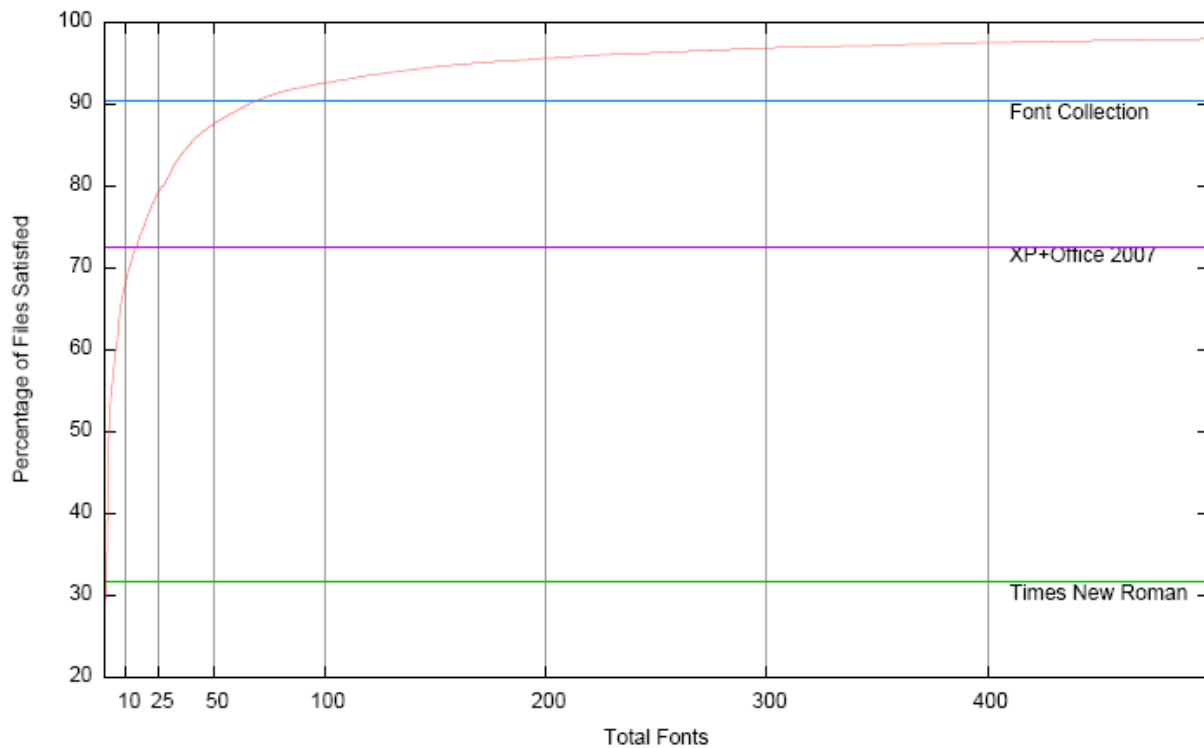


Figure 4: Font Usage for Government Documents

Default	Do not care or don't know	42%
Roman	Serif fonts with variable stroke width	28%
Swiss	San serif fonts with variable stroke width	21.5%
Modern	Constant stroke width	4.5%
Script	Handwriting	2%
Decorative	Novelty or other	2%

Table 4: Font Family

As mentioned, many fonts are used for a single or few glyphs. In the case of the document collections studied, nearly 10% of all referenced fonts were never used for more than a single glyph of these. Unfortunately, even assuming all these fonts are known (by adding the names to our database) doesn't appreciably alter our "satisfaction" rate.

Font Substitution

Locating missing fonts using name strings as a primary identifier is challenging. While a number of websites providing font search (one-at-a-time) based upon

large databases, they are often incomplete, inaccurate, and cannot be automatically queried. We encountered a variety of problems with names containing non-ASCII characters, compressed names, variations of known names, and rare or specialized fonts. Processing compressed names in particular may be platform specific. For example, the Windows fonts named "Helv" and "Tms Rmn" in Windows 3.0 were renamed to "MS Sans Serif" and "MS Serif", respectively, in Windows 3.1. Modern iterations of the Microsoft operating system maintain a system-accessible map in the font substitution registry subkey to correctly map these names. However, on Macintosh platforms the proper substitutions remain unavailable.

Font substitution in Microsoft Windows is performed according to a "closest match" criteria calculated as a weighted sum from a vector of information corresponding to the LOGFONT structure. As the name suggests, this information does not correspond to a "physical" font (e.g. the data that defined a font installed or loaded into the task environment) but rather a "logical" font composed of a sequence of properties requested as a result of a user action or application request during the loading of a document.

When an application is tasked with displaying text in a given font, it performs an API function call populated with values from the LOGFONT corresponding to the desired (logical) attributes. Windows realizes a best match

to the desired font by searching the installed fonts to find one with values closest to this attribute set. As previously noted, these attributes include font family names, font weight, font width, and font slope. An obvious consequence of this is that the (absolute) best match may correspond to a font not installed on the system.

Font matching information is precalculated and cached from existing font collections according to a detailed algorithm designed to support efficient font matching on request. In addition to the attributes noted above, this process includes the generation of combined Family and Face names, extraction and resolution of various terms for style and weight (for example, “Bold Face” to be treated the same as “Bold”), and the extraction of canonical numerical representations.

Fonts are subsequently matched by location of an exact (or longest-substring match) FontFamily name, a matching face from the candidate face list (computed as a weighted attribute vector based on FontStretch, FontStyle, and FontWeight - prioritized in that order), and localization settings.

Microsoft provides detailed information on font matching in Windows under the Windows Presentation Foundation (Microsoft 2009d). In some cases, a matched font may not contain the required glyphs (for example, a font with a Latin-only glyph-set requested to render glyphs in an unsupported codepoint range). The Microsoft Unicode rendering service, Uniscribe, will automatically render the unsupported script in the appropriate fallback font (Kaplan 2009).

In order to provide more accurate mapping from logical to physical fonts and account for improved rendering technologies, Microsoft introduced improved APIs in Windows XP (moving from GDI to GDI+) and Windows 7 (with the development of DirectWrite). As a result, the font substitution actions performed by different versions of Microsoft Office may be inconsistent. Additionally, the default font substitution algorithm implemented in GDI+ may be overridden within an application. Because of this, it can be difficult to accurately replicate substitution actions performed by a proprietary application such as Microsoft Word. This can be demonstrated via a simple experiment using fonts provided with the operating system. If a LOGFONT structure is populated completely with information from a font loaded into the system font table using an API call such as AddFontResourceEx(), and that font is then unloaded, a subsequent call to the function CreateFont() under Windows XP (GDI+) will frequently result in mapping that differs from the substitution performed by Microsoft Office.

Furthering the problem, there is not enough information included in a legacy Word (.doc) file for any font used to completely fill the LOGFONT structure. Font information that is available, such as the PANOSE number or a substitution “hint”, cannot be passed directly

to font mapping functions within GDI+. This is important because it exposes a significant preservation issue; even when document specifications are well documented (or fully open), the behavior of the application most commonly used to render those documents becomes the defacto standard for all rendering services. If the behavior of such an application depends on code or API functions that are not generally exposed, the “openness” of the format is not necessarily a guarantee of preservation-friendliness.

David Levy notes that the severity of the risk posed by font substitution depends on a variety of factors, that “even in these simplest of cases, sensitivity to the circumstances of use is crucial to determining what is to be preserved—what counts as successful preservation” (Levy 1998). Font selection algorithms are complex, and the substitution actions performed may be opaque to anyone who is not a typographer or software developer. However, simple tools can assist the user in determining the degree to which a rendered document is well-formed (or exhibits information loss). In the following section, we briefly describe a prototype to provide this service.

Discussion

We have shown that the majority of digital documents obtained from a wide range of sources (up to 79%) can be rendered accurately using fonts appearing in modern desktop environments such as the combination of Microsoft Windows and Microsoft Office. With a small amount of additional work—using information drawn from font foundries, performing family name matches for legacy fonts or commercial fonts for which distribution has ceased, we can expect to increase this coverage to 92%.

This nevertheless leaves a large number of documents unaccounted for. Microsoft’s own search engine indexes nearly 60 million documents currently available on the web. At this level of coverage, 1.8 million documents are guaranteed to be rendered inconsistently on a typical workstation. For many of these documents, the loss of information may be negligible. It is impossible, however, to quantify this without appropriate software tools to analyze the risk to a particular collection.

Even with access to the full documentation for a legacy proprietary format, replicating the behavior of the original environment used to render that document can be extremely difficult—or impossible—for tasks as seemingly basic as font selection and use. While publishers and institutional archives build support around work-flows optimized for “born archival” documents, the majority of the documents produced in the world today continue to be created using proprietary office software with font embedding disabled. With an open syntactic

specification for the documents, the font identification and selection problem remains.

Automated tools to simplify the identification and location of missing fonts in document sets can significantly reduce the risk of information loss in an archive. As part of our ongoing research, we are developing tools to assist archivists in processing and analyzing font information from large collections of documents. Our current tool is capable of automatically preprocessing collections of Microsoft Word documents, identifying ranges of characters for which fonts are not available, and rendering those ranges separately from the remainder of the document for examination. In concert with this tool, we have developed a plug-in for Word which automatically locates and highlights ranges for which fonts or characters have been substituted.

Our research demonstrates the need for simple, effective tools to correctly identify font information and locate missing font data in order to facilitate lossless rendering. We show that effective rendering of heterogeneous document collections can only occur when supported by a database of information drawn from multiple vendors; no existing identification technology provides universal or even adequate coverage. Proper archival handling of these digital objects should include tools to rapidly and selectively present to a human relevant document segments for quality assurance in order to mitigate risk during subsequent archival and access events.

References

[Adobe Developer Connection 2009] 2009. Font technical notes. <http://www.adobe.com/devnet/font>

[Bauermeister 1987] Bauermeister, B. 1987. *A Manual of Comparative Typography*. Van Nostrand Reinhold.

[Impson 2005] Impson, J. 2005. Evaluating the IBM and HP/PANOSE font classification systems. *Online Information Review* 29(5):14.

[Kaplan 2009] Kaplan, M. 2009. Font substitution and linking #1. <http://blogs.msdn.com/-michkap/archive/2005/03/20/-399322.aspx>

[Lachowicz 2009] Lachowicz, D. 2009. wvWare, library for converting Word documents. <http://wvware.sourceforge.net/>

[Levy 1998] Levy, D. M. 1998. Heroic measures: reflections on the possibility and purpose of digital preservation. In *DL '98: Proceedings of the third ACM conference on Digital libraries*, 152–161. New York, NY, USA: ACM.

[Microsoft Developer Network 2009] 2009. DirectWrite API Reference. [http://msdn.microsoft.com/enus/library/dd368038\(VS.85\).aspx](http://msdn.microsoft.com/enus/library/dd368038(VS.85).aspx)

[Microsoft 2009a] 2009a. List of fonts supplied with Microsoft products. <http://www.microsoft.com/typography/fonts/product.aspx?PID=1>

[Microsoft 2009b] Microsoft. 2009b. LOGFONT. <http://msdn.microsoft.com/en-us/library/ms901140.aspx>

[Microsoft 2009c] 2009c. Microsoft Word 2007 Binary Format Specification. [http://download.microsoft.com/download/0/B/E/0BE8BD-D7-E5E8-422A-ABFD.../Word97-2007BinaryFileFormat\(doc\)Specification.pdf](http://download.microsoft.com/download/0/B/E/0BE8BD-D7-E5E8-422A-ABFD.../Word97-2007BinaryFileFormat(doc)Specification.pdf)

[Microsoft 2009d] 2009d. WPF font selection model. <http://blogs.msdn.com/text/attachment/2249036.ashx>

[Reichherzer and Brown 2006] Reichherzer, T., and Brown, G. 2006. Quantifying software requirements for supporting archived office documents using emulation. In *JCDL '06: Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, 86–94. New York, NY, USA: ACM.