

Fast Optical Character Recognition through Glyph Hashing for Document Conversion

Kumar Chellapilla
Microsoft Research
kumarc@microsoft.com

Patrice Simard
Microsoft Research
patrice@microsoft.com

Radoslav Nickolov
Microsoft Research
radonick@microsoft.com

Abstract

This paper proposes a glyph hashing approach to optical character recognition with applications in document conversion. The viability and efficiency of the approach is tested through its implementation in a print driver on 68,987 PDF documents containing 1.15 billion characters. Results indicate that a hash table with (a) 3.2 million hashes is sufficient to represent all characters from these documents, and (b) 480 fonts are sufficient to cover over 90% of these documents. Glyph recognizing experiments indicate that 80% of unique character glyphs and over 96% of all characters from unseen documents can be found in a hash table built using all 68,987 documents.

The hashing approach is used to not only recognize the character codes but also, size, style (bold, italic, etc), and font name. We found that the hashing approach can scale to hundreds of fonts and thousands of characters per font. Further, it is extremely fast and can recognize over 100,000 characters per second. Owing to its speed, such a hashing approach can complement any existing OCR system by acting as a pre-filter to produce a 4-5 times speedup during document conversion.

1. Introduction

More than 90% of the documents in the world are created using word processing software. However, several competing document formats exist and offer a variety of trade-offs. Importing documents into any particular word processor has traditionally relied on custom converters that could read and convert between these formats. Such converters are time consuming, expensive to build, and are brittle to changes in document formats. In this paper, we present an approach for automated OCR in a low-level print or display driver towards the goal of building an automated document conversion/import system that can work with any document that can be printed or viewed.

2. Background

Data hashing techniques are well known for their use with hash tables. Hash tables provide efficient constant time lookup of data [1]. Hashing of images in both vector and raster formats for search has recently gained much attention. Schemes for approximate and perfect matching using image hashes have become available and are used in the retrieval of images from image databases [2-4].

Optical character recognition (OCR) of print characters is a well studied problem. Print character OCR is of two types: a) Type 1: the character glyphs are slightly distorted or modified during successive presentations, b) Type 2: the character glyphs are identical over independent presentations.

The first class is well known and is usually what is referred to when one mentions OCR. Common examples include OCR for print, scan, fax, and digital camera images. Each of these images undergoes modifications due to filtering and noise during the acquisition process. As a result, even the state of the art OCR systems have the following drawbacks: a) They do not guarantee 100% accuracy (even after rejection) b) They are slow (typical systems can process 1000 characters per second) c) They do not scale to a large number of fonts, styles, and character sets.

The second class of OCR problems wherein the character glyph is identical over independent presentations may offer special potential for overcoming these drawbacks. Further, the use of OCR in certain niche applications such as document conversion requires that these drawbacks be overcome for their success.

In this paper, we investigate the viability and success of such a hashing based OCR for use in document conversion systems. A hashing approach is presented for recognizing character glyphs to obtain their character code (Unicode value), font size, font name, font attributes (italics, bold, etc) and any other font and character metadata that is of interest. The

proposed approach is (a) very fast with constant time lookup, (b) close to 100% accurate (after rejection), (c) and can scale to thousands of fonts, font styles, and character sets with several thousand characters (even to all unique Unicode character glyphs).

3. Method

Our approach takes advantage of the fact that when software renders characters to a device (display, print, etc) the generated characters are identical or have a few limited variations. This facilitates building a very fast OCR based on glyph hashing. Any viable data hashing scheme can be used. Examples include Universal Hashing [1], Message Digest (MD2, MD4, MD5), Secure Hash (SHA-1, or its 256, 384, and 512 bit variants), etc [5-6]. Each of these hashing algorithms takes a variable length sequence of bytes and returns a fixed length hash. This hash is used as a unique value to represent the character glyph data. Hashes of two glyphs match if the corresponding glyph data also matches. Small changes to the data result in large, unpredictable changes in the hash [1].

3.1. Glyph Hashing

Character glyphs can be hashed using either their contours or rasterized bitmaps. Glyph hashes can be collected in a hash table and used for OCR. The size of the table can be chosen for an optimal trade-off between available memory and desired classification accuracy. The proposed glyph hashing system has the following features:

- a) Hash functions for hashing glyphs in bitmap and/or contour formats
- b) A procedure for building a large glyph hash table
- c) A labeling approach for verifying and filling in missing information about these glyphs.

3.1.1. Contour hashing. When glyph contours are hashed, only the control points of the associated Bezier or Cardinal splines need to be hashed. The advantage of contour hashing is that control points for contours are independent of font size. Thus the resulting hash values are independent of font size. This can produce significant space savings and allows for a smaller set of hashes for any font. However, the contour data stored as floating point values can result in lookup failures due to round-off effects. Further, contour hashing requires that the contours be available during retrieval.

3.1.2. Bitmap hashing. On the other hand, glyph bitmaps can be directly hashed. While most font

formats allow access to glyphs in both contour and bitmap formats, some fonts are available only in rasterized formats (aptly called raster fonts). Raster fonts have different bitmaps for different font sizes. For raster fonts bitmap hashing is the only viable approach.

3.2. Font Recognition

When documents are printed/viewed the associated print/display driver gets a collection of font objects followed by several characters to be printed/displayed. These characters are identified by associated glyph indices into collections of glyphs present in the font objects. Many document processing applications embed document fonts directly into the document during creation. This allows for these documents to be displayed and printed with visual fidelity even on machines that do not contain fonts used in the document. During printing/viewing, the print/display driver is provided a temporary font object, which is little more than a set of glyphs. The extra metadata might be insufficient to determine the font's name, style, size, or character code (ASCII or UNICODE) etc. Using a font hash approach outlined here, one can hash and lookup each of the glyphs in the temp font. Using the retrieved font information one can efficiently and with high confidence determine the temporary font name and its characteristics.

3.2.1. Ambiguous Glyphs: Even though the mapping from a glyph to the glyph hash is not one-to-one, the probability of collision is very small. For instance, if we hash 10 million glyphs with a 64 bit hash function, the probability of any glyph colliding with another is $10^7/2^{64}$ or less than one chance in 10^{13} (be it in contour or bitmap formats). Indeed while we indexed millions of characters, the only collisions that we detected were between identical bitmaps (they were not hash collisions). For example, in the well known Helvetica font the lower case L ('l') and the upper case I ('I') have the same glyphs, and as a result hash to the same value. Similarly, the underscore character ('_') and hyphen character ('-') are identical across some fonts, and periods are identical across many fonts. Making font name decisions based on all glyphs in the font allows for their ambiguity to be resolved.

3.2.2. Glyph Hash Caching: One advantage of using font recognition in a device driver based document converter is that though a document might have several thousand or even hundred thousand characters, the number of fonts in the document is relatively small, typically less than 10 (see Figure 3). Thus, each (temp

font, glyph index) pair need only be looked up once in the large glyph hash table and cached to recognize every instance of the associated character. Caching provides over an order of magnitude speedup during document conversion. It also makes loading the large hash table in memory unnecessary, allowing for faster start time.

3.3. Building a Glyph Hash Table

One procedure for building a large glyph hash table is the following:

a) Collect a large corpus of frequently used documents obtained through the use of free internet search engines (such as Google, MSN Search, Yahoo, etc) and a web crawler. Glyphs from fonts commonly have the same structure independent of which document format or software they come from.

b) Through a print or display driver extract all temp fonts and associated glyphs. All one needs is software for viewing or printing these documents.

c) Label these glyphs either using a minimal parser (that can parse for character and font information). In many cases a character/font parser is much simpler than the associated document authoring/viewing software. Also, one need only be able to parse for characters present in the document. In a) collection can be limited to documents that can be easily parsed. Labeling can be skipped if the glyphs being labeled are from fonts already installed on the system. In such cases, the print driver can obtain true font information.

d) Compute bitmap / contour hashes for these labeled glyphs and build a glyph hash table.

This procedure generates a hash table with a natural distribution of the hashes. Alternative approaches for populating such a hash table are also possible. For example, the corpus of documents from step a) can be replaced with a large collection of fonts. In this case, the generated hash table will contain a uniform distribution of hashes for each font.

The glyph bitmap hashing, font recognition, and glyph hash caching procedures described above were implemented as part of a print driver for use in a commercial Office suite of applications. Since PDF (portable document format) is the most widely used document format, 68,987 publicly available PDF documents (in English) were collected through the use of a web crawler. These were printed and the glyph hashes were collected using the print driver. Along with the hashes, glyph size, glyph locations (page and character bounding boxes), and some limited style information was also collected. A simple PDF parser

capable of extracting words, word bounding boxes, and font names was used for labeling.

4. Results

A total of 1.15 billion characters from 68,987 PDF files were processed through the print driver to build a hash table containing 3.2 million hashes. The print resolution was 300 dpi, and the hash values had 64 bits. More than half of these unique hashes had less than 4 samples, which indicates a very skewed distribution.

4.1. Character, Hash, and Font distributions

The number of characters, unique hashes, and font styles in the documents are presented in Figures 1, 2, and 3, respectively. For each curve, the documents have been sorted by corresponding quantity. Figure 1 illustrates the high variability in document size. The documents were sampled from the web and were already sorted by relevance by a search engine, so they are fairly representative. Figure 2 and 3 show that the distributions of hashes and fonts are very skewed. Due to the large variation in the document parameters of interest, both mean and median parameter values are presented in parentheses as (mean,median) pairs. The average document had (21K, 8K) characters and produced (315, 239) unique hashes that span (4.76, 4.0) font styles. Since glyph bitmaps were hashed, both size (8pt, 12pt, etc) and style (italic, bold, etc) were used in counting the number of font styles. The (21K, 8K) character count puts the average document size between 3 and 9 pages (with 2500-5000 chars/page). It is interesting to note that on average less than 350 unique hashes need to be recognized to OCR the whole document! Even to cover 99% of the observed documents, we need no more than 1200 hash lookups. Note that these hashes are recognized only once and are cached for performance. These numbers indicate that though the total number of possible hashes is large (3.2 million) the size of the live hash table needed to process a single document can be quite small.

4.2. Character coverage

Two methods were used to gauge what fraction of the characters from a random document would be found in the hash table for varying hash table size. The first approach was to examine the natural distribution of characters and their hashes. The second approach looked at predicting character distributions in unseen documents. Results from the second approach are presented in section 4.4.

Figure 4 shows the percentage of characters (over all documents) found as a function of the hash table size. While as little as 100 hashes are sufficient to cover 25% of the characters, about 75,000 characters are needed for 90% character coverage. All 1.15 billion characters collapse to a hash table with 3.2 million entries. A hash table with 3.2 million entries takes up less than a few megabytes of memory and can be compactly stored on today's computers. Figure 5 presents the distribution of unique hashes over the ASCII character range of Latin characters in the range of 33-126. Character ranges of interest 0-9, a-z, and A-Z are indicated on the x-axis. The observed distribution reflects a scaled version of the natural distribution of characters in the English language. As expected a-z make up the largest fraction followed by A-Z and 0-9.

4.3. Font coverage

The hash table building procedure used in this paper relied on sampling a large corpus of documents. However, a similar hash table can be built by collecting a large number of fonts and hashing glyphs contained in them. One can enumerate all possible font sizes and styles of interest to produce a hash table with a uniform distribution of character codes. Drawbacks of this approach are the need for fonts on multiple platforms, printer and/or postscript fonts, and a font parser for different font types (true type, type1, type 2, etc). Further, the hash table size explodes quickly with no easy way of inferring which hashes are important (occur more frequently) and which ones are not.

One can use a combined approach wherein font encountered in the corpus of documents are separately collected and completely hashed (at all font sizes and style combinations of interest). We did not choose this approach as we were interested in a natural distribution of fonts in real documents. However, we can estimate how effective such an approach would be by counting the number of font styles covered as a function of the hash table size. Results are presented in Figure 6. Note that different font sizes are treated as different font styles. From Figure 6, about 4000 font styles are sufficient to cover 90% of the collected 3.2 million hashes. These 4000 font styles result in about 488 unique fonts. Thus 90% of the fonts can be covered if we hash these 488 unique fonts.

4.4. Character coverage in unseen documents

The font hash lookup is only as useful as its size, variety, and applicability of the glyphs it contains. In a real-world application, no matter how big your hash

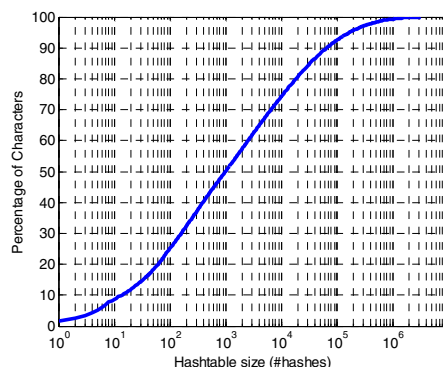


Figure 4. Character coverage with growing hash table size. 1.15 billion characters gave 3.2M hashes.

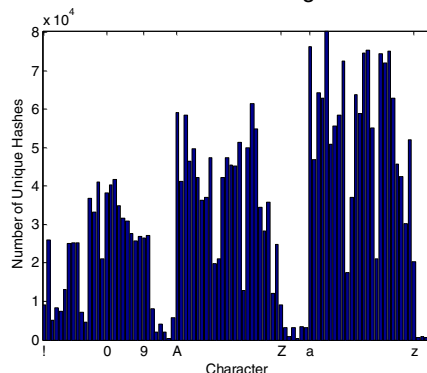


Figure 5. Distribution of hashes (ASCII: 33-126).

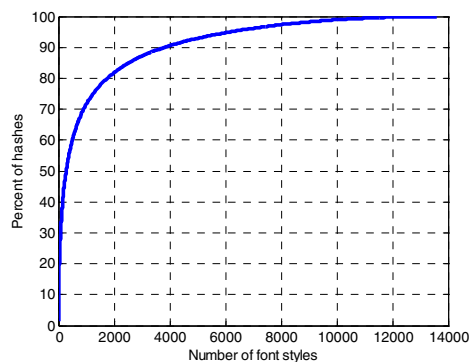


Figure 6. Font style coverage with growing number of styles hashed.

table, you will encounter documents with characters that are not found in the hash table. In such scenarios, a traditional OCR (which is slower) system can be used to obtain missing information. We estimate how often you can expect to find entries in the hash table (on unseen documents) by using only a fraction of the collected documents for building a hash table (seen documents) and testing the built hash table to OCR the remaining documents (unseen documents). Figure 7 presents results from these experiments for varying

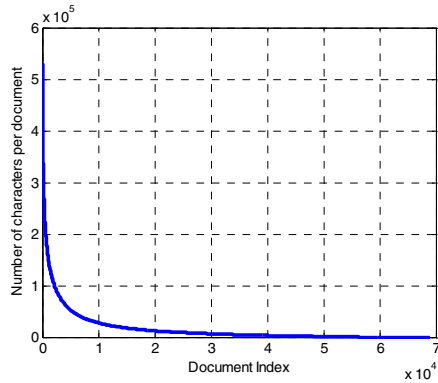


Figure 1. Number of characters/document (sorted).

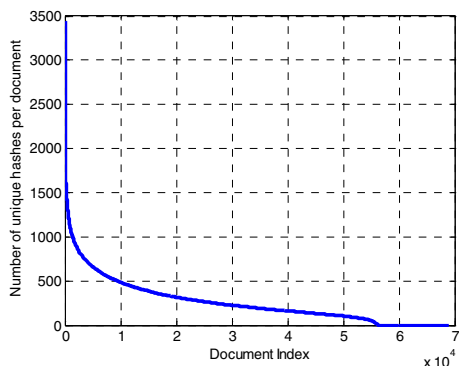


Figure 2. Number of hashes/document (sorted).

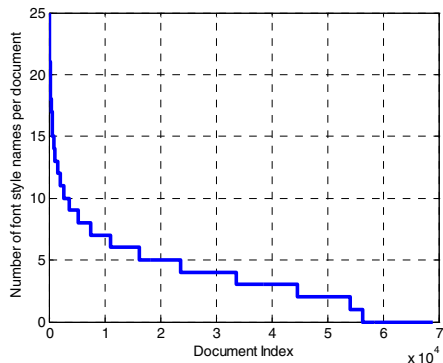


Figure 3. Number of font styles per document (sorted).

percentage of seen documents (used to build the hash table). With 63K documents (90% of all documents used in this study) being hashed, we successfully predict about 70% of the hashes in the remaining 7K unseen documents. The missed hash entries belong to infrequent characters. As a result, 70% of the hashes found in the hash table account for over 96% of the characters in these 7K documents. With all 70K documents hashed, we expect to be able to predict about 80% of character glyphs and over 97% of all characters in unseen documents.

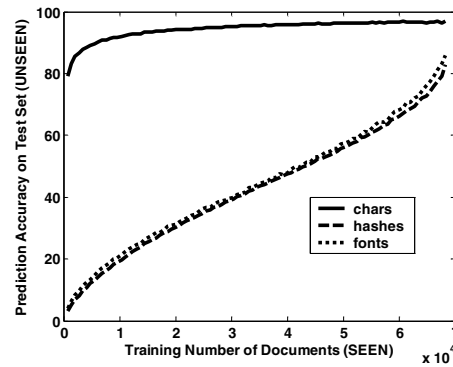


Figure 7. Percentage of glyphs (hashes), characters, and fonts in unseen documents that can be found in the hash table built using seen documents. Seen documents are used to build a hash table that is used to OCR characters in unseen documents. Mean results over 10 independent trials are presented.

5. Conclusion

In this paper, we presented a fast OCR system based on glyph hashing for recognizing characters during document import/conversion. This system was implemented in a print driver and tested on 68,987 PDF documents. A hash table with 3.2 million unique entries is sufficient to hold all 1.15 billion characters from these documents. This clearly shows that such a hashing based approach is not only viable but also effective for character recognition. Further, such a system is very fast with the current implementation being capable of hashing over 100,000 characters per second on a P4 3GHz machine. Experiments with finding glyphs in unseen documents showed that about 80-90% of unique character glyphs from unseen documents can be found in the hash table.

6. References

- [1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. *Introduction to Algorithms*, MIT Press, 1990.
- [2] MJ Fonseca, B. Barroso, P. Ribeiro, JA. Jorge, "Retrieving ClipArt Images by Content," *Intl. Conf. on Img. and Video Retr. (CIVR'04)*, Ireland, Jul. 2004.
- [3] BB Kimia, "Shape Representation for Image Retrieval," in *Image Databases*, John Wiley & Sons, 2002.
- [4] E.G. M. Petrakis and C. Faloutsos, "Similarity Searching in Large Image Databases," Technical Report 3388, Department of Computer Science, Univ. of Maryland, 1995.
- [5] R. Rivest, "RFC 1321 - The MD5 Message-Digest Algorithm," MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.
- [6] D. Eastlake, 3rd, and P. Jones, "RFC 3174 - US Secure Hash Algorithm 1 (SHA1)," Motorola (Eastlake) and Cisco Systems (Jones), Sep. 2001.