Presented at IS&T/SPIE EI'97, Conference 3027: Document Recognition IV
Feb 12-13, 1997, San Jose, CA.

Embedding Digital Data on Paper in Iconic Text

Dan S. Bloomberg

Xerox Palo Alto Research Center
Palo Alto, CA 94304

ABSTRACT

A system has been built that embeds arbitrary digital data in an iconic representation of a text image. For encoding, a page image containing text is analyzed for the text regions. A highly reduced image of the page is generated, with an iconic version of the text that encodes an input data stream substituting for the text regions. The data is encoded into modulations of rectangular iconic representations of text words, where the length, height and vertical positioning of rectangles, as well as the spacing between rectangles, can all be independently varied. No correspondence need be maintained between the words in the document and the word icons. Word icons or other marks on each line can be used for identifying, calibrating and justifying iconic text. Decoding proceeds by finding iconic lines and determining the iconic word sizes and locations. Word icons printed with 8x reduction are reliably decoded from 300 ppi binary scans.

One application is to present iconified first pages of many documents on a sheet of paper, where the URL of each document is encoded in its icon. Icon scanning and selection then allows retrieval of the full document. Another use is to print an icon on every page of a document, containing meta-information about the document or the specific page, such as the version, revision history, keywords, authorization, or a signed hashing of the full image for authentication.

Keywords: embedded digital data, iconic, watermarking, image segmentation, document imaging, page segmentation, image reduction

1 Introduction

With the ease of electronic printing has come a proliferation of paper documents that coexist with electronic versions. Additionally, there are legacy and other paper documents for which the electronic versions are not available. Document storage and retrieval systems put various requirements on the ability to interconvert between paper and electronic forms. For example, for some systems it is useful to place machine-readable data on the printed document that refers to one or more electronic versions.

Machine-readable data can be placed directly on a document in a variety of ways, which can be classified by their degree of visibility. Barcodes, either 1D or 2D, are visible on inspection as machine-readable data. Watermarks are at the other end of the spectrum; the data is hidden within the elements of the document that would appear virtually

the same in its absence. Watermarks are particularly difficult to place on printed text. Brassil et al. showed that a small amount of data could be hidden in printed text using either small displacements in the vertical locations of text lines or in the horizontal locations of text words within each line.[5] In between these extremes, data can be placed within a special graphic element in such a way that it is not obvious on casual inspection. An example is Xerox DataGlyphs,[4] which have the appearance of a uniformly stippled region, particularly when the marks (typically ellipses oriented at ± 45 deg) are separated by less than 0.020 inches. The purpose of this paper is to present another example of inconspicuous data embedded in a graphic element, where the graphic element may have an additional recognition function for the observer.

Thumbnail reductions of page images have been used in bitmap displays, to allow selection based on our ability to identify different documents (or classes of documents) by inspection.[13] Peairs has described the use of "paper icons" for allowing a machine to distinguish between reduced images (*thumbnails*) of printed pages, so that a selected thumbnail could be used to refer unambiguously to its originating document.[11] He chose to produce an *iconic* representation of the page by replacing the text of each thumbnail by a sequence of tiny black rectangles, where each rectangle corresponded to a character in the original image. This *icon* accurately preserved the appearance of the full resolution image, and the number of characters in each word could be determined from a high resolution scan of the icon. This string of numbers is a signature that can be mapped into an index in a database of documents.

The iconic representation used by Peairs bears a closer resemblence to the thumbnails from which it is derived than is required for visual identification of the page. The most significant visual clues are derived from the overall layout, embedded figures and line-art, and large text. However, at a scan resolution of about 40 ppi, text in 14 pt font is not readable when printed at about 8x reduction on a 300 ppi printer. Consequently, we are at liberty to make larger changes in the small text regions of the icon, and we do this for two purposes: to embed *arbitrary* digital data in the text regions and to enable reliable readback of this data from medium resolution (e.g., 300 ppi) binary scans.

The appearance of an iconic image is relatively unchanged if, in the regions of small text (say, less than 14 pt), the pixels corresponding to each text word are replaced by a solid rectangle that fills the bounding box for that word. We refer to this substitution as "word greeking". In order to embed arbitrary digital data within these text icon regions, we propose to further alter the iconic image in a number of ways that preserve the visual appearance of greeked text. Namely, we use greeked rectangles of various locations and shapes within each iconic text line that are chosen specifically to encode arbitrary digital data.

The data within the icons will be referred to as *blocks*. Although the shapes of the blocks roughly approximate the distribution of word sizes that would appear at the thumbnail resolution, *no attempt is made to correlate block shapes with word sizes occurring at the same location in the thumbnail.*

Before proceeding, we step back and consider iconic representations of documents more generally, distinguishing between two different types: *quasi* representations and *stylistic* representations. Quasi iconic representations, such as those generated by Peairs, present sufficient similarity to the thumbnail to make them visually recognizable as a stand-in for it. Stylistic iconic representations are similar to icons appearing on graphical user interfaces, in that they represent a *class* of documents, and they may contain readable text to distinguish between members of the class. For example, a stylistic representation may have the title in a font large enough to read, under which the digital data is written using lines of word-shaped blocks, as discussed above. In the following we consider only quasi iconic representations.

There are many applications in which techniques for embedding arbitrary information unobtrusively within an image can be combined with the use of an iconic image representation, in order to both encode useful information in an indiscernible manner and to make use of human pattern-matching abilities. For example, the iconic image is a useful mechanism for inconspicuously embedding digital information in images in any application where the

presence of an iconic image is provided as a surrogate for a full-sized version of an image. Thus Peairs' application can be implemented by placing the URL of the document within the icon, eliminating the need for a database of word-length signatures for each document. This and other applications are described in Sec. 4.

1.1 Plan of the paper

Two processes, encoding and decoding, are described in Sec. 2 and Sec. 3, respectively. The encoding process requires segmentation to locate the regions of normal sized text that will be replaced by data blocks in the icon. The mapping from data to blocks must be done in such a way that it mimics the appearance of text words, and several methods are described that obey this constraint. The decoding process also begins with segmentation to locate the blocks within the icon. This is followed by measurements on the blocks to determine first the method used for encoding the message in the blocks. Then the measured parameters of the blocks are used to reconstruct the message. This is followed in Sec. 4 by examples of document imaging applications that are enabled by iconic data.

2 Encoding of Iconic Data

To encode an arbitrary message within an iconic image of a page, the regions of median-sized text must be located and ordered in some arbitrary but non-ambiguous way. The digital data may be altered in several ways to allow recovery and improve the appearance of the icon. A method is then chosen to map the message into data blocks arranged within the text parts of the icon.

2.1 Image Segmentation in Encoding

If an electronic representation of the document (other than a raster image) is available, segmentation can be performed either directly on that representation, or on a raster image generated from it. For the former, generically useful segmenters would operate on either a high-level markup language (e.g., SGML) or on one of the common and powerful page description language representations of the resulting image, such as PDF or PostScript. Doerman and Furuta described a page segmenter using TeX's DVI format.[6] A PostScript and PDL parser that extracts ASCII text, *pstotext*, is publicly available from DEC[7] in their Virtual Paper project. The most generic segmenter works directly from the raster image, and that is the type used here.

The encoder segmentation operations that are required form a subset of the layout analysis used for document image summarization,[3] with some changes in the textblock sieving and ordering. The major operations required are (1) image deskew,[2] (2) identification and removal of halftone images,[1] (3) identification of regions that consist of text blocks, (4) selection of text blocks with fonts in a prescribed range of sizes, and (5) ordering the text blocks. In the process, the font size and interline spacing is measured for each selected text block.

In step (4), we first measure the average inter-line spacing and the font size, and test against a criterion that the text is not readable when printed out at 8x reduction. This criterion is that the font is smaller than approximately 14 pt. The text blocks with predominant font size and inter-line spacings, and with a minimum of 3 text lines, are selected. In step (5) the text blocks are ordered in strict column-major order, rather than row-major order, to reduce ambiguity. Subsets are grouped into "columns" that all mutually overlap in the horizontal direction. A block that acts as a horizontal bridge between blocks with no horizontal overlap is assigned to the left-most column. Within

size	*x-height*	*max vert extension*	*char width*	*inter-line*
10 pt	3	5.5	2.5	7
12 pt	4	7	3	9

Table 1: *The approximate size, in pixels at 7x reduction and 300 ppi, of font-related quantities for Times Roman: x-height, maximum vertical extension including ascenders and descenders, average character width and inter-line spacing.*

each column, blocks are ordered vertically from top to bottom, and then the columns are ordered from left to right.

If the blocks are to be laid out over each text line, the appearance of the paragraphs will automatically be maintained. With a more free-form layout, it may be desired to replicate in the icon the type of paragraph layout that appears in the original. A simple heuristic is used to determine the paragraph layout method. In each text block, check the location of the beginning and ending of each text line, relative to the lines above and below and to the left and right edges of the text blocks. It is easy to determine if most of the right edges are aligned (justified). For short lines that are not right justified, if the following line is indented the paragraph formatting is likely to be by indentation. Otherwise, if the following line follows at a larger vertical spacing the paragraph formatting is probably indicated by extra vertical apace.

2.2 Layout of Iconic Data

The key requirements of iconic data encoding are that (1) the data regions in the icon have the visual appearance of words, (2) the data has a characteristic appearance that allows the decoder segmenter to find it reliably, and (3) the data itself is quantized so that it can be reliably decoded from a 300 ppi scan of a 300 ppi laser printer.

The encoder segmenter finds the approximate character heights and line spacings within textblocks that are chosen for embedding data. Text acceptable for iconic replacement at approximately 7x reduction has font sizes not greater than 14 pt, and line spacing up to about 28 pt. The decoder segmenter's ability to group separate lines into a "text block" places the upper limit on line spacing. Table 1 gives approximate sizes, in pixels, of 10 and 12 pt Times Roman text at 43 ppi (of the original image), corresponding to about 7x reduction on a 300 ppi scan.

The various block sizes and line spacings in the icon should be chosen to give an appearance similar to the text in the thumbnail. The actual text may be both left and right justified, and may delineate paragraphs by first line indentation or extra inter-line spacing. The degree to which such details are emulated in the icon blocks is application-dependent. Right justification may not be necessary visually, but it is relatively easy to achieve by several methods, at a cost of some loss in ability to encode data:

- Use the white space between the blocks.
- Use the first or last block to justify the lines, and do not encode data in its length.
- Use variable length marker blocks at one or both ends of each line to provide both right justification and vertical alignment (See Sec. 2.4).

Of these three, the use of white space for justification is generally the most satisfactory, both visually and in data-carrying capacity, and this method will be illustrated here. If emulation of paragraph locations, with or without the paragraph delineation method used in the original, is not required, then the location of paragraphs and their visual representation may be chosen for simplicity and reliability of decoding.

Two different methods of formatting the blocks have been used. In the *line-preserving* method, the appearance of the original text formatting is maintained with respect to line location and line length. Only the lengths of the blocks within the line differ from the original words. This method works for all textblocks, and is required when the text is not well block-structured, as when it fills around an irregular object (see, e.g., Fig. 1). In the *free-formatting* method, no attempt is made to conform to line lengths or locations. The inter-line spacing of the original text can be used as a guide in choosing the number of data block lines, but this is not necessary. Paragraph delineation is arbitrary.

The encoding methods can be chosen either statically or dynamically. Different encoding methods have different appearances, and some may be chosen in advance because they are preferable for an application. The encoder can also choose the method dynamically, depending on the font size and line spacing used. For example, if when encoding with the *line-preserving* format, there is more latitude to vary the block height for larger text fonts. Thus, 14 pt fonts may be encoded with three different heights, 11 pt and 12 pt with two, and 10 pt or smaller with uniform block height. The choice should be done at the granularity of the text block, and equal sized fonts should use the same encoding methods, so that the decoder can unambiguously determine the encoding method for each text block.

When the blocks are formatted using *line-preserving*, or to simulate paragraphs, short lines will occur. If the first or last block is not used for line justification, then we use the rule that data is always stored on a line, even one consisting of a single block. In this case, since the length of the block is determined by the data and not the size of the original line, there may be some discrepancy between the two sizes, even when using the *line-preserving* method.

2.3 Encoding data in block and space lengths

There are several methods for encoding data in the symbol lines. First, it should be noted that there are up to four data channels, which can be used either independently or in combination. Data in these channels is determined by

- The length of the block symbols.
- The length of the space between block symbols.
- The location of the top edge of the symbols.
- The location of the bottom edge of the symbols.

Each of these elements must be quantized in such a way that the data is preserved through the distortions of printing and scanning. The size of the quantization is determined by the scanning quality. Table 2 gives suggested values for the minimum quantization increments for the four channels, when using either binary (1 bpp) or grayscale (4 or 8 bpp) scans at 300 ppi. When decoding the channels, both lengths and edge positions are found by averaging over the block. The quantization of the top and bottom edges can be smaller than that of the block lengths. This is because the top and bottom edges are usually much longer than the side edges, and the error in determing edge

Image Type	*Block Length*	*Space Length*	*Top/Bot Edge*	*Block Height*
binary	3	2	2	2
grayscale	2	2	1	1

Table 2: *Minimum quantization values for channels, for binary and gray scans.*

Block Lengths	*Space Lengths*	*Code-rate*
(1,4)	1	0.57
(1,4)	(1,2)	0.75
(2,5)	1	0.44
(2,5)	(1,2)	0.60
(1,8)	1	0.55
(1,8)	(1,2)	0.67
(2,9)	1	0.46
(2,9)	(1,2)	0.57

Table 3: *Typical code rates for different block and space parameters*

position varies inversely with the edge length. To reach these minimum values, the shorter block symbols should not be coded using the top and bottom edge information.

We consider two different methods for encoding data in the length of the blocks, and similar techniques also apply for encoding data in spaces between blocks. We refer to the unit of length quantization as the *message unit*, and the *code rate* is the number of data bits that can be encoded by each message bit.

The first method, *block-by-block*, encodes a fixed amount of data in each block, regardless of the length of the block. The data encoded on each line will not be constant. Each block can take one of a set of N different lengths (e.g., from 2 to 9 units). Where N is a power of 2, each block can directly encode an integer number of bits. However, N need not be a power of 2. For example, if N = 6, then each pair of blocks (36 different states) can encode 5 bits, with 4 illegal states that can be used for error detection. For random data, all block lengths can be expected to occur with equal frequency. Table 3 gives the code rates for blocks, for the cases where N = 4 (e.g., $w_{min} = 2, w_{max} = 5$) and N = 8 (e.g., $w_{min} = 2, w_{max} = 9$). The spaces are of width 1 or 2 message units, corresponding to the ability to store 0 or 1 data bits, respectively. For simplicity, rendering units of blocks and spaces are of equal length, and the code rate is then the average number of data bits per rendering unit.

More generally, the block and space sizes are described by three parameters: a minimum size, a quantized increment, and the number of different lengths that can be used. When spaces are used both to pad the block lines and to carry data, an ambiguity arises when the lengths of all the spaces on a line are equal: it is not possible for the decoder to determine which of the data bits are represented by the spaces. A simple resolution of this problem is that for lines where the encoded spaces are all identical, no data is placed in the spaces. (At the encoder, this requires re-encoding the line).

The second method for encoding data in block lengths, *run-length-limited,* is analogous to RLL encoding used in magnetic recording, where constraints are placed on the minimum and maximum number of message bits between a transition.[8] Here, we place a constraint on the minimum and maximum block lengths. The code rate is constant, so

the amount of information encoded on each line depends only on the line length and is independent of the data. As an example, the 1,7,2,3,1 RLL code has a single table for implementation, requires a minimum length of 1 message unit, a maximum of 7, and has a code rate of 2/3, encoding 2 data bits in each 3 message bits. Including the white space between blocks, the actual code rate is less than 2/3, and is data dependent.

Fig. 1 shows icons for a typical set of first pages of documents, at a reduction of 6x. The encoding method is block-by-block, with 3 bits/block encoded in the block length and 1/bit encoded in each space on lines whenever not all spaces are the same length. The icons are arranged in order of increasing stored data capacity, in bytes: {79, 144, 149, 191, 199, 207, 215, 238, 314, 355, 356, 380}.

2.4 Encoding data in top and bottom block edges

When data is encoded in the vertical location of the top and bottom edge channels, there are two situations to be distinguished, depending on the necessity to register the edges in absolute vertical position.

In the special case where the block heights differ for all possible encodings, it is sufficient to measure the block height only. This has the advantage that vertical edge registration is not required, and it makes the decoding easier. For example, 1 bit can be encoded in the block by having a nominal position for a "0" bit, and moving either one or both edges in such a way that the block height changes for a "1" bit. The measurement of block height can be accurately made from a binary scan when the block is encoded with 1 pixel difference in heights, for blocks that are longer than about 8 pixels. Alternatively, word shape can be mimicked by using blocks that are in one of three states, each having different block height: nominal height, "ascender" only, and "ascender" + "descender". Then two adjacent blocks can be used together to encode 3 bits.

Where data is encoded in the absolute position of the top and bottom edges of the blocks, it is necessary to provide reference marks for determining these positions. One method is to use the top and bottom edges of the first and last blocks in each line for registration. Symbols between these references can have their top and bottom edges moved up or down independently from the reference line. Alternatively, the first and last symbols in each line can all be of the same or similar size, to provide line-by-line verification to the decode segmenter that the line contains data symbols. For example, each data line can begin and end with a small symbol that has similar visual appearance to a small word, is easily identified by the decoder, is not a typical shape for symbols representing data, and can provide registration data for the line. However, this has the disadvantages of uniform appearance, and is not typically required for accurate segmentation.

2.5 Use of multiple channels

As an example of the use of multiple channels, consider the following configuration:

- The block lengths vary between 1 and 8 units, and the dat is encoded block-by-block (3 bits).
- The spaces have two different lengths (1 bit), except for the first space which is the shorter length and sets the reference size. Otherwise, the spaces are padded equally for left and right line justification.
- The top edge edge has two positions (1 bit), a nominal "x-height" and a raised height, except for the first and last blocks.

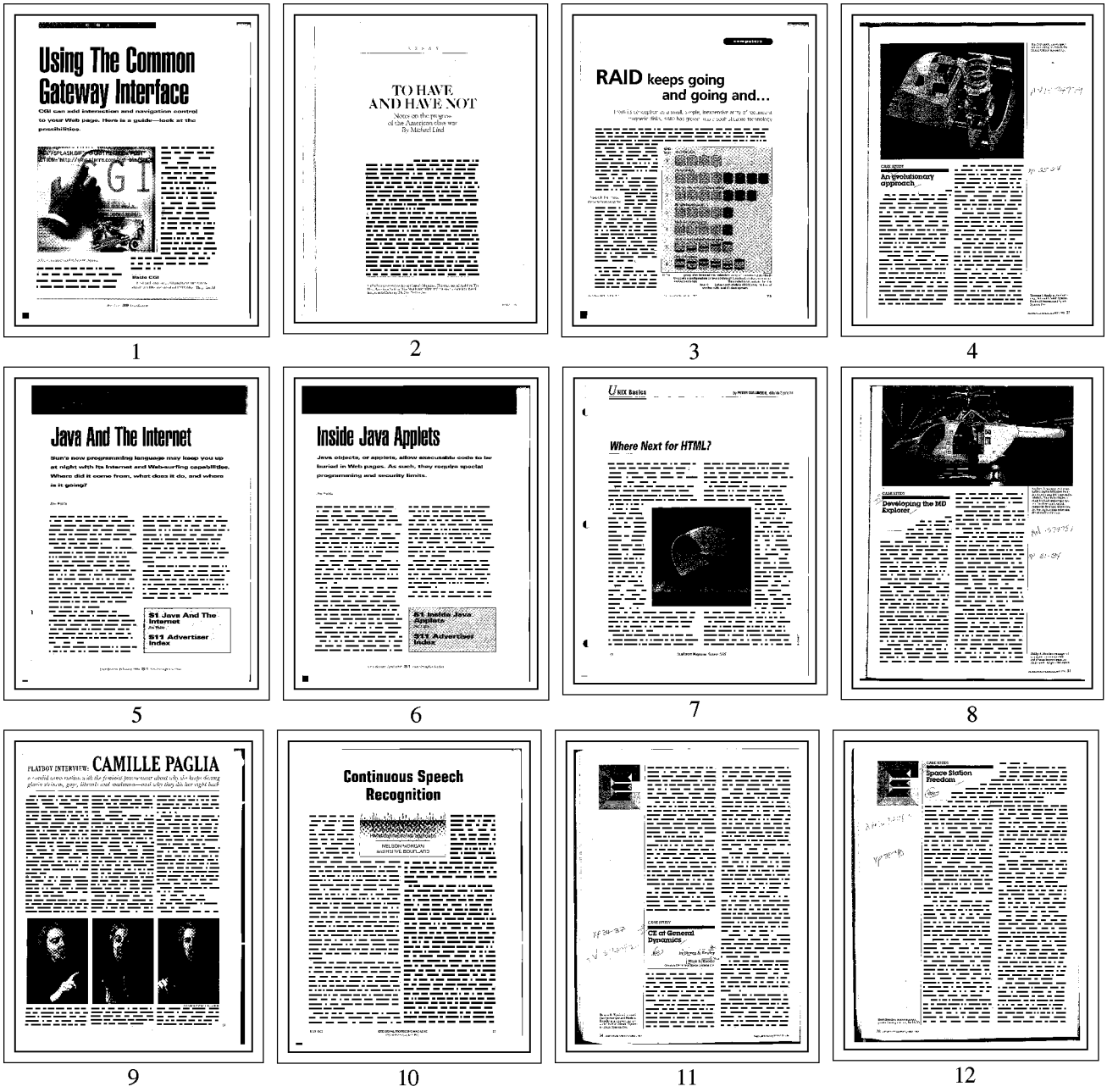

Figure 1: *Icons with block length and space length encoding, at 6x reduction.*

- The bottom edge edge has two positions (1 bit), a nominal baseline and a lowered height, except for the first and last blocks.

The space padding in each line is determined by setting the minimum space and the constant space increment; it is then distributed equally among all spaces. With these parameters, the first and last blocks encode 3 bits, all other blocks encode 5 bits, and all spaces but the first encode 1 bit. The average block length is 4.5 units and the average space is 1.5 units, plus padding. For long lines with many blocks and little padding, the net code rate thus approaches an upper limit of 1 bit per message unit.

In Fig. 2 we show four different encodings at 6x reduction, using the *line-preserving* method because of the irregular text columns in the original image. In all four icons, 3 bits of data are encoded in the block length. Additionally, in the second icon, 1 bit of data is encoded in each space; in the third, 1 bit of data is placed in the block height; and in the fourth, both block height and space are used to encode 2 bits. Fig. 3 shows, at 7x reduction, the thumbnail and four icons with the same encoding sequence. Similar results at 8x and 9x reduction are given in Fig. 4 and Fig. 5, respectively.

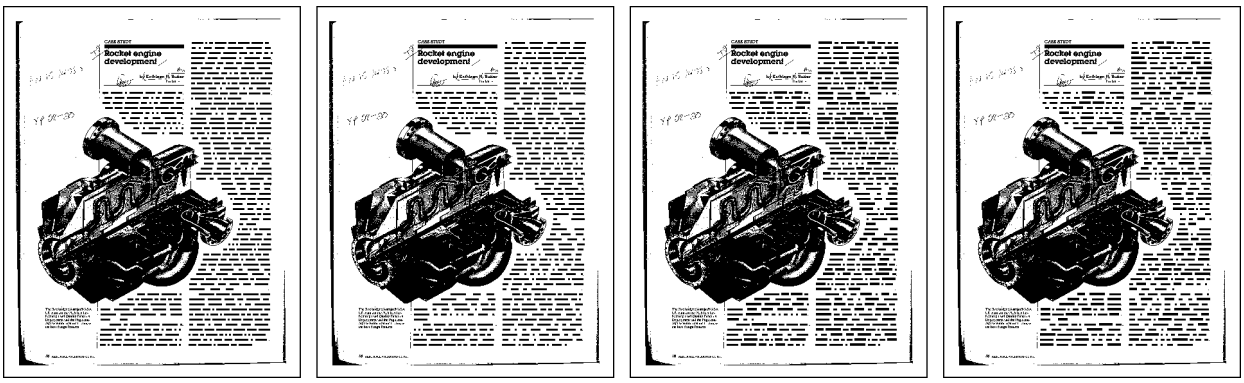

Figure 2: *Icons with different encodings at 6x reduction. From left to right, data is encoded in B, BW, BV and BWV, where B = black run lengths, W = white spaces and V = top and bottom edges of black runs.*

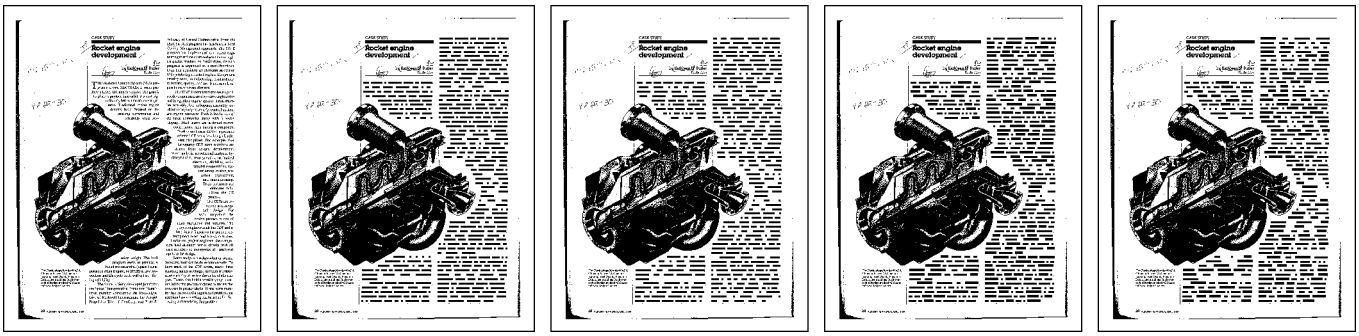

Figure 3: *Thumbnail and icons with different encodings at 7x reduction.*

Table 4 gives the number of bytes of random data that are stored in each of these icons. The amount of data decreases approximately *linearly* with reduction, rather than as the square, because an attempt is made to scale the block sizes and spacing with the icon reduction. The linear decrease in density comes from two factors. First, the scaling is limited by minimum sizes for quantization values (see Table 2), which reduce the number of blocks that can be placed on each line. This number is further increased because the lines are short, and with fewer blocks

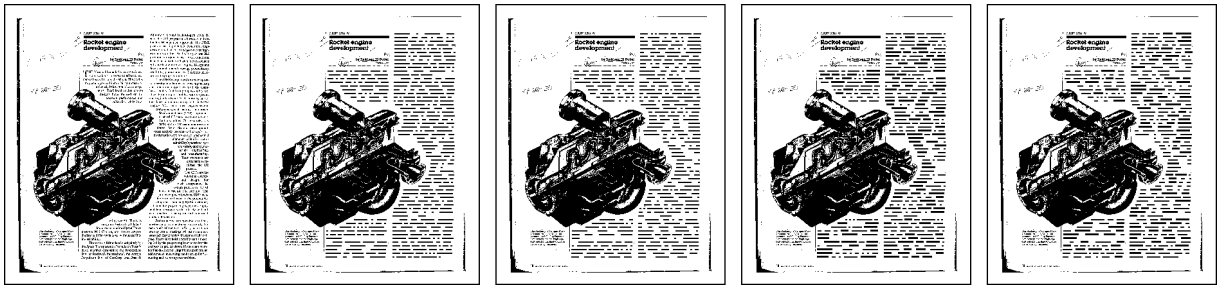

Figure 4: *Thumbnail and icons with different encodings at 8x reduction.*

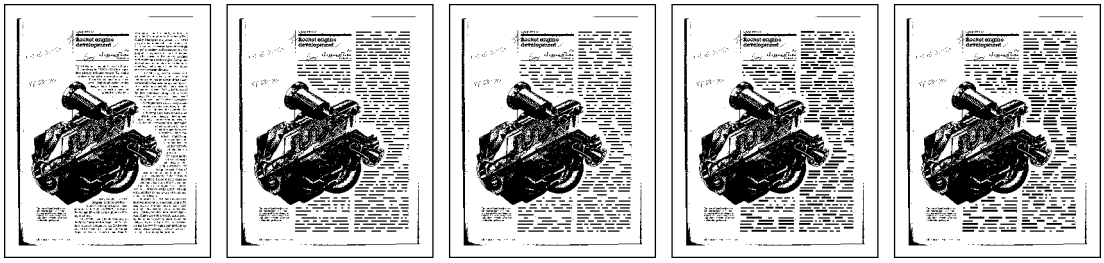

Figure 5: *Thumbnail and icons with different encodings at 9x reduction.*

a proportionately larger fraction is taken by white space padding to justify both edges of each icon line. With the parameters used, putting data in white spaces or the block heights adds about 20 - 25 percent to the baseline storage in the block lengths alone. Block height was not encoded on the two (of eight) shortest blocks. When both block height and spaces are used to store data, the increase is smaller than the sum of increases when either are used separately because the use of white space decreases the number of blocks. This effect is more pronounced for images such as those shown here, where the text lines are short and we are justifying each one to the original text line.

2.6 Encoding metadata

In any encoding situation, one has to consider what information about the data or its encoding needs to be provided to the decoder. There are three types of such metadata: information about the message, the encoded data stream, and the rendering into imaged blocks.

Reduction	*B*	*B,W*	*B,V*	*B,W,V*
6x	186	227	218	262
7x	157	187	190	209
8x	138	165	158	189
9x	120	144	144	160

Table 4: *The number of bytes of random data stored in the icons in Figs 2 - Fig 5. Data is encoded in combinations of B (black run lengths), W (white spaces), and V (black run heights).*

Typically, the number of bytes in the message is pre-pended to the message. This allows the addition of random arbitrary data to be added to the message so that it fills the iconic text regions. If the message is restricted to strings, it can be encoded with a null termination before the added random data, thus avoiding the need for including the length explicitly. If the message can be arbitrary digital data, it is advisable to XOR encrypt the message using a 32-bit word with roughly equal numbers of 0s and 1s. This randomizes the message and removes situations where there are possibly long strings of 0 or 1 bits. The same word is used for decoding, restoring the message to its original form.

The message is typically encoded to correct errors that might occur in the decoding process. Block encoding, such as Reed-Solomon, is often used when the errors are expected to occur in bursts.[12] The error correction parameters are either known to the decoder *a priori*, can be guessed by the decoder by trying different combinations, or are placed *unencoded* at the beginning of the message. In some cases, a small cyclic-redundancy checksum (CRC) is appended to the message to check the error correction. The most serious error is loss of synchronization, and the only remedy is to limit propagation. This can be done in some situations by using special codes or layout conditions to separate encoded blocks.

The decoder must also have information about how the data is encoded into the four rendering channels. Inspection of the blocks yields easy determination of the minimum and maximum block lengths, the block length quantization, and whether or not the white space and edge channels are being used. Further, the specific use of the edge channels can be inferred from the number of different states that are observed. The only problem is determining if the first and/or last blocks are used for line justification, rather than data, which requires a priori information at the decoder. This is another reason for not using the blocks for line justification: that is the only situation for which it is necessary to include metadata on the channel encoding.

2.7 Distortions introduced by scanning

Fig. 6 demonstrates the problem measuring block height from a binary scan. This is a segment of icon data printed at 8x reduction. Both the print and scan resolutions were 300 ppi. Because of skew on scanning, the scanned and printed pixels go in and out of vertical alignment in approximately vertical bands. The skew angle in radians is the inverse of the number of horizontal pixels between bands. All blocks were printed 3 pixels in height. When vertically aligned, the scanned blocks have smooth edges, but when out of alignment the edges are noisy and the average height varies from 3. Because the average height can be less than 2.5 or greater than 3.5, the block height increment used for coding binary images must be greater than 1, as is given in Table 2. For grayscale scans, the block height is easily resolved to unit pixel increments.

3 Decoding of Iconic Data

There is an interesting performance asymmetry between the encoding and decoding segmenters. The encoding segmenter has more variable input data, but if it misses some text regions that would have been fair candidates for embedding data, the system doesn't fail; it just becomes less efficient in data storage. On the other hand, the decoding segmenter is searching for the icon data blocks. These are highly regular image components and relatively easy to locate, but it must reliably find *all* of them.

One effective method for locating the data blocks in the icon is to find the connected components (4-connected is satisfactory). Each component is examined to determine if it has characteristics of a possible data block. Both the

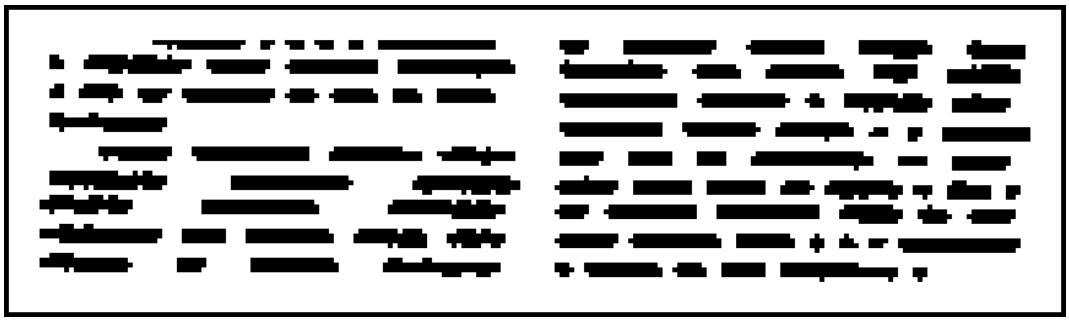

Figure 6: *Typical binary scan of icon at 8x reduction, showing effect of pixel quantization and scan skew on horizontal edges. Note the alternating bands of well-formed and poorly-formed blocks.*

length and height must be a minimum of 2 pixels. Further, the bounding box for the connected component must be relatively full with ON pixels. The pixels on the boundary of the bounding box can be noisy, but a very large fraction of the pixels interior to the bounding rows and columns should be ON. A filtered image is then constructed of those connected components that satisfy these tests. The selection threshold should be set to get all the true blocks, and this allows a small number of noise components that must be removed.

As with the encoding segmenter, we use morphological operations to help locate the ”text blocks” in the icon. A combination of horizontal and vertical closings will join the data blocks. Long vertical runs of OFF pixels are identified and used to prevent joining columns. Once the ”text blocks” are isolated, the connected components can be individually tested for size, shape, and location within regular data lines. In this way, the blocks are ordered and noise is removed.

All the data blocks are next measured to determine the location of their edges. For example, to measure the vertical location of the top edge, the top pixels are located exclusive of those at the left and right edges of the bounding box. From the measurements, a histogram of block and space sizes is constructed for each ”text block”, and the type of quantization, number of levels, and best sizes for each quantization level are determined.

Using these quantization levels, the data is then extracted from each data block. Any data near a quantization boundary is flagged as uncertain, and can be handled by erasures in the error correction encoding. After decoding the message and XORing with the same word used for encryption, the CRC is inspected to determine that the message was correctly decoded.

4 Applications

We briefly consider three applications, each of which makes special use of the iconic form. The first and most obvious application is similar to that of Peairs, where the paper icon represents the actual document, e.g., through encoding the URL. If a single icon is printed on a page, the electronic document can be automatically selected when the page is scanned. If a page contains more than one icon, a means must be provided for the user to indicate to the system which icon is to be selected.

The second set of applications is to print an icon on every page of a document, containing meta-information referring to either the document or the specific page. For the former, the icon can hold the document version,

revision history, authorization, or keywords.

The icon can also be used for authentication, by inserting a digitally-signed hash code, possibly including the sender's public key. The hash function can be derived from the text if the electronic form of the document is available. However, even if only a scanned image is available, a hash code can be derived from it by a method similar to that used by O'Gorman and Rabinovitch for photographic images.[10] That method requires dividing the image into N x N pixel squares, and deriving a number for each square that depends on the relative average gray values between that square and its neighbors. The relative value must be used because although absolute gray values can change due to printing, copying and scanning, the relative values tend to be more stable. For text, it is more appropriate to use text-like features, such as the number of edge pixels at various orientations, rather than photographic features such as average gray values. These text-like features tend to be stable when copying and scanning, so it is possible to develop from them a number for each square that depends on the feature values in that square. From these text-based features, the hash code for the image can be developed. Where there is a mixture of image and text on the page, a segmenter can be used to determine which features (or none) should be used for each grid. One special feature of the icons is that they *don't even require a a hashing function on the image*, because the icon itself has the approximatey appearance of the page. The signature can encode a known text (to verify authorship) and a person can visually inspect the page and the icon to verify that they are similar.

A third application is to provide information that would enhance the performance of a character recognition system. If the original document is in electronic form, then the actual text is known at the time the iconic image is constructed. Lopresti and Sandberg described a method for generating error-correction data that is specially suited for text..[9] For each line of text, they generate a decimated string consisting of one bit for each character, plus a hash string for error detection. This requires about 20 percent of the full ascii text, which is too much for the iconic encoding described here. However, we are able to store iconic data that contains information about the number of words in a line or the number of characters in each word. Further, by encoding the data in each iconic line *to correspond with a text line in the original image*, we can register the data explicitly with the source text. The word length information can do more than identify recognition errors. Many errors are due to merged characters, and the *a priori* information about the number of characters in a word can help during recognition. Word length information can also be used to reduce the number of split or joined words. These methods can be used even if the original image is not in electronic form, because the image can be scanned at high resolution in grayscale to provide a very accurate measurement of the text properties. This data, when encoded in the icon, is then used to improve OCR derived from medium-resolution binary scans.

5 Summary

Arbitrary digital data can be embedded inconspicuously in an iconic representation of a text image. The method described here encodes the message into the shape and location of rectangular blocks that give the visual appearance of text at high reduction. For medium resolution printers and binary scanners, more than 100 bytes of data can reliably be encoded on an icon of a typical page image. The actual density is about 250 bytes/in^2 on a 6x reduced image and 350 bytes/in^2 on a 8x reduced image. With grayscale scanners, the quantization parameters can be reduced, allowing about 50 percent more data to be stored.

The ability to embed arbitrary data enables a number of document imaging applications that link the paper and electronic worlds. The machine-readable data can refer to the location of the electronic document or to metadata relating to it, to the specific paper document carrying the icon, or to electronic data in the document itself. Examples in the first case are the URL, version and keywords; in the second case printing, authorization and authentication

information; and in the third case key numerical data, the location and description of key components such as section headings and electronic backings (e.g., spreadsheets) for tables, and information about the text that would improve OCR.

6 REFERENCES

[1] D. S. Bloomberg, "Multiresolution morphological analysis of document images," *SPIE Conf. 1818, Visual Communications and Image Processing '92*, pp. 648-662, Boston, MA, Nov 18-20, 1992.

[2] D. S. Bloomberg, G. E. Kopec and L. Dasari, "Measuring document image skew and orientation," *SPIE Conf. 2422, Document Recognition II*, pp. 302-316, San Jose, CA, Feb 6-7, 1995.

[3] D. S. Bloomberg and F. R. Chen, "Extraction of text-related features for condensing image documents," *SPIE Conf. 2660, Document Recognition III*, pp. 72-88, San Jose, CA, Jan 29-30, 1996.

[4] D. S. Bloomberg, "Binary image processing for decoding self-clocking glyph shaped codes," U.S. Patent 5,168,147, Dec. 1, 1992.

[5] J. Brassil, S. Low, N. Maxemchuk and L. O'Gorman, "Electronic marking and identification techniques to discourage document copying," *IEEE Journal on Selected Areas in Communications*, vol. 13(8), pp. 1495-1504, October 1995.

[6] D. S. Doerman and R. Furuta, "Image based typographic analysis of documents," *ICDAR '93*, pp. 769-773, Tsukuba, Japan, Oct. 20-22, 1993.

[7] "pstotext," http://www.research.digital.com/SRC/virtualpaper/pstotext.html.

[8] P. A. Franaszek, "Sequence-state methods for run-length-limited coding," *IBM J. Res. Dev.* **14**, pp. 376-383, July 1970.

[9] D. P. Lopresti and J. S. Sandberg, "Certifiable optical character recognition," *ICDAR '93*, pp. 432-435, Tsukuba, Japan, Oct. 20-22, 1993.

[10] L. O'Gorman and I. Rabinovitch, "Photo-image authentication by pattern recognition and cryptography," *Int. Conf. Pattern Recog. '96*, pp. 949-953, Vienna, Aug. 25-29, 1996.

[11] M. Peairs, "Iconic paper," *ICDAR '95*, pp. 1174-1179, Montreal, Quebec, Aug. 14-16, 1995.

[12] W. W. Peterson and E. J. Weldon, *Error Correcting Codes*, 2nd ed, Ch. 5, MIT Press, Cambridge MA, 1972.

[13] G. A. Story, L. O'Gorman, D. Fox, L. L. Schaper and H. V. Jagadish, "The Right-Pages image-based electronic library for alerting and browsing," *Computer*, pp. 17-26, Sept. 1992.