*Steven C. Bagley and Gary E. Kopec*

# *Editing* Images of Text

Recent advances in image scanning, storage, and retrieval have stimulated interest in incorporating scanned documents within electronic document systems. Integrating scanned documents with structured documents is an important step toward making electronic document processing universally available. Most approaches to this problem are based on one of two paradigms—bitmap editing and format conversion.

Conventional bitmap editors and paint programs treat scanned images as simple pixel arrays without internal structure. A major strength of this approach is that bitmap editors are applicable to an open-ended class of images. Moreover, almost by definition they preserve the detailed format and typographic design of the input material. In particular, if distortions due to scanning and printing are ignored, a bitmap editor behaves as an identity system if no editing operations are performed.

There are numerous scenarios in which editing an image while preserving the appearance of unedited material is important. In general, such scenarios involve documents that originate in image form and are to be retained in image form after modification. Examples include last-minute correction of spelling mistakes before photocopying, modifying viewgraphs at meetings, exchange of document drafts by fax, and recreational forgery (the construction of obvious parodies for humorous purposes). While it would seem that bitmap editors are ideally suited to such applications, in practice their utility is often limited.

The primary problem is that current bitmap editors support only relatively low-level editing operations. Typical facilities include selection of polygonal regions; cutting, pasting and copying selected regions; and painting operations such as bit setting, clearing, and complementation. No attempt is made to classify the content of the image (e.g., as text or line art) and no operations are provided that assist higher-level content-specific operations. Thus, for example, using a paint program to delete a character from the middle of a word might

involve selecting the character by drawing a polygon around it, clearing the selected region and then selecting and moving the following characters to close the gap. As an isolated operation, this sequence of actions is not overly complex. However, text editing at such a low level quickly becomes tedious. As an example, consider the case of right-justified text where deleting a character requires readjusting all the interword spaces of the line and possibly moving words from one line to another.

Format conversion aims to convert a scanned document image into a structured electronic representation that can be manipulated using conventional types of editors. Recognition procedures such as page segmentation, character recognition, and raster-to-vector conversion are used to produce a document description in a language, such as SGML or ODA, that is similar to the descriptions used to create documents electronically [2]. The recovered symbolic description is intended to completely capture the content and format of the document and to be used in lieu of the scanned image for all subsequent document processing.

Format conversion is an appropriate approach when significant use will be made of the resulting symbolic document representation. Examples of such applications include transferring a large amount of material from one document to another, language translation, and full-text database entry. For scenarios involving simple image-to-image transformations, however, current format conversion technology has several problems.

The first problem concerns current recognition algorithms, which exhibit neither the accuracy nor the robustness for totally autonomous operation. As a result, format conversion typically involves a significant amount of manual proofreading to correct recognition errors. A recent estimate of the total cost for conversion, including optical character recognition (OCR) and proofreading, is between $2 and $5 per page [13]. The effort required to proofread a long or complex document may be excessive if the ultimate goal is to perform a small amount of relatively simple editing.

The second problem is that although format conversion can successfully capture the content and logical structure of a document, it is less likely to preserve distinctive typographic details such as hyphenation, line breaks, leading, and fonts. As previously suggested, a desirable property for an editor of image-based material is that it function as an identity system unless the user explicitly carries out editing operations. The state of the art in document recognition is unlikely to satisfy this criterion for the foreseeable future.

Another problem is that the symbolic description generated by format conversion is often expressed in a language that is idiosyncratic to a particular document processing system. Because there is no universally accepted interchange standard for compound documents, using a structured document representation can actually restrict rather than extend the availability of a document.

Thus we see that conventional bitmap editors are applicable to an open-ended class of images and preserve image appearance but typically have no recognition capability that might assist higher-level operations such as text editing. Conversely, systems based on format conversion create symbolic descriptions that greatly facilitate structured document processing tasks, but are typically limited in the range of document images they will handle reliably. Bitmap editors and conventional document recognition systems are therefore at opposite ends of a spectrum parameterized by increasing operation complexity and decreasing image generality.

This article describes Image EMACS, a text editor for scanned document images, which illustrates an intermediate point between the bitmap editing and format conversion paradigms. The aim of Image EMACS is to provide familiar high-level editing commands in a context that preserves document appearance except where the commands are actually applied.

The inputs and outputs of Image EMACS are binary images of text. The primary document representation within Image EMACS is a collection of image elements extracted through simple geometrical analysis, rather than an abstract symbolic description of the image. Image EMACS is based on a principle of *minimal interpretation:* perform image analysis only to the extent necessary to carry out operations explicitly requested by the user. Adherence to this principle avoids the format conversion problems previously noted. Our approach also builds on existing networks and protocols for communicating document images and provides a natural and smoothly evolving path from paper-based work flows toward the ubiquitous use and interchange of electronic documents.

Image EMACS can be viewed as an extreme form of a WYSIWYG (What You See Is What You Get) page-composition system. As such, it provides two distinct, but related, classes of editing operations. The first class is based on viewing text as a linear sequence of characters. The linear sequence model forms the basis for common text-editing operations such as character insertion and deletion, string search, and linear cursor movement. The second class of operations is based on viewing text as a two-dimensional arrangement of glyphs on an image plane. Operations of the second class are concerned with aspects of typography such as character and line spacing and line justification.

The linear text-editing operations of Image EMACS are patterned after those of the text editor EMACS [14]. In terms of linear text editing, users of Image EMACS should succumb to the illusion that they are using EMACS to edit an ordinary text file, displayed as lines of characters, although Image EMACS is actually manipulating image components extracted from a bitmap. The choice of EMACS in this respect is not essential and a variety of other editing paradigms could have been used. To emphasize this point (and for the benefit of readers not familiar with EMACS) we will describe Image EMACS commands generically using descriptive names and will minimize references to particular user interface or keyboard interaction styles.

The typographic facilities of Image

XEROX operates a private long-distance telephone system called Intelnet. The network consists of 12 interconnected AT&T telephone switching centres with circuits to every major location in the United States, Europe, Canada and Mexico. Intelnet, when properly used, can be an important productivity tool, providing faster, more convenient and less expensive telphone service than the public long-distance networks.

EMACS are not currently patterned after any particular system. Many aspects of typographic structure take the form of global constraints on the spatial arrangement of text elements. Detecting and enforcing such constraints is computationally expensive, and the development of appropriate algorithms to do so is a topic of current research in the document recognition community. For this reason, the typographic facilities of Image EMACs are only those necessary to support linear text editing and to imitate certain simple capabilities found in text editors such as EMACS.

### Linear Text Editing in Image EMACS

Text editing in Image EMACS is based on two observations: Many text editing operations do not make use of the character labels of the characters being edited (such as during copying or deleting); and a character in an English "book" style typeface is usually realized as a single connected region of black pixels (a connected component [5]). The correspondence between characters and connected components is well known and is exploited in many character recognition systems. The central insight behind Image EMACS is that *many text editing operations can be implemented directly in terms of geometrical operations on connected components, without explicit knowledge of the symbolic character labels, that is, without character recognition.*

In many text editors, editing operations are performed on a *buffer* containing a set of linked lines, where each line consists of a sequence of characters. Image EMACS constructs a similar data structure from an image of text by first segmenting the image into a sequence of lines and then performing connected component analysis on each line. Line segmentation is currently based on the assumption that lines are separated by at least one complete row of blank (i.e., white or '0'-valued) pixels. Figure 1 shows a sample text image with a bounding rectangle drawn around each character. (The terms *character* and *connected component* will frequently be used interchangeably, and in place of the more precise description "a group of connected components that probably looks like a character.")

There are several important situations in which the correspondence between characters and connected components is more complicated than a simple one-to-one mapping. A single connected component in an image, for example, can be the image of several characters if the characters are blurred together or form ligatures (e.g.,'fi' and 'ffi'). Figure 1 contains an example of merged characters, the 'as' in 'faster' on the seventh line. Conversely, several connected components can belong to the same character if parts of the character have dropped out or the character intrinsically includes several parts (e.g., 'i,' ';'). Finally, formatting "characters" such as Space and CR are usually not manifested as printed glyphs at all.

**Figure 1.** Connected component analysis of a sample text image. A bounding rectangle is shown around each set of connected components that have been grouped into a character-like unit.

Image EMACS handles multipart characters with simple connected component grouping rules based on similarity of horizontal position of component bounding boxes. These rules seem to correctly handle 'i', 'j', and all punctuation symbols except for double quotation marks. Figure 1 contains several examples of 'i' and 'j.'

The current implementation does nothing special about ligatures and treats them as single characters. We have found the atomicity of ligatures to be only a minor nuisance. Editing most often involves deleting, copying, or moving all of the characters in a ligature anyway. In other cases it is simply necessary to delete the ligature and retype one or more of the individual components.

If the horizontal distance between two consecutive components on a line is larger than a threshold, the inter-component space is considered to

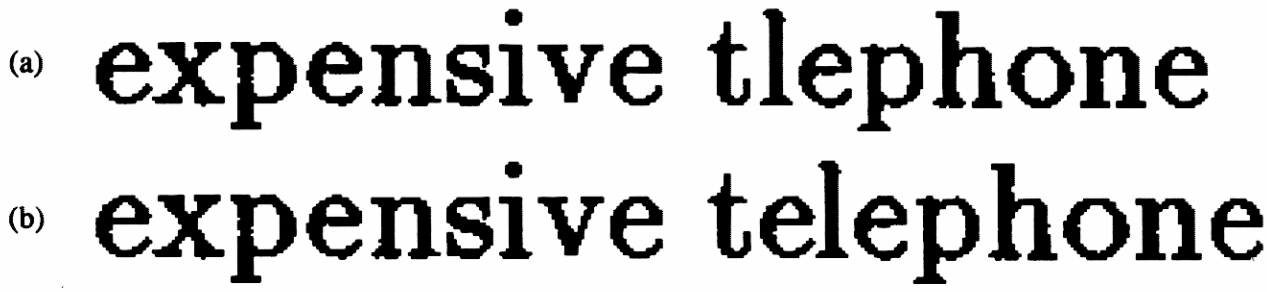



**Figure 2.** Example use of copy-mouse-char to correct a spelling mistake. A copy of the first "e" in "expensive" is inserted after the "t" in "tlephone": (a) Before; (b) After.

correspond to a real Space character and thus to indicate a word boundary. For simplicity, each interword space is treated as a single Space character, independent of its size. Each line is assumed to be terminated by a CR character.

Table 1 identifies a small subset of the Image EMACS linear text editing commands. Most editing operations take place in the vicinity of a distinguished buffer position that is marked by a cursor. The cursor movement commands identified in Table 1 adjust the position of the cursor. The insertion and deletion commands provide various forms of "cut and paste" editing. For example, the command cut-region removes the characters between the cursor and a second buffer position called the *mark*. Cut text is stored in a special buffer from which it can be pasted back, either at the same position (to undo the cut) or at a different position (to move the cut text). The command copy-mouse-char inserts at the cursor a copy of the character located under the mouse. Figure 2 illustrates a simple use of copy-mouse-char to correct a spelling mistake in the text of Figure 1.

Inserting characters by normal typing requires having established associations (bindings) between the keyboard keys and the character images. The key binding commands in Table 1 create and manipulate key bindings for printing characters. The command key-bindings-from-font binds keys to character images taken from a stored font. When key bindings are set from a font, documents produced using Image EMACS look very much like those from a conventional word processor, except they are represented as bitmaps rather than as sequences of character codes.

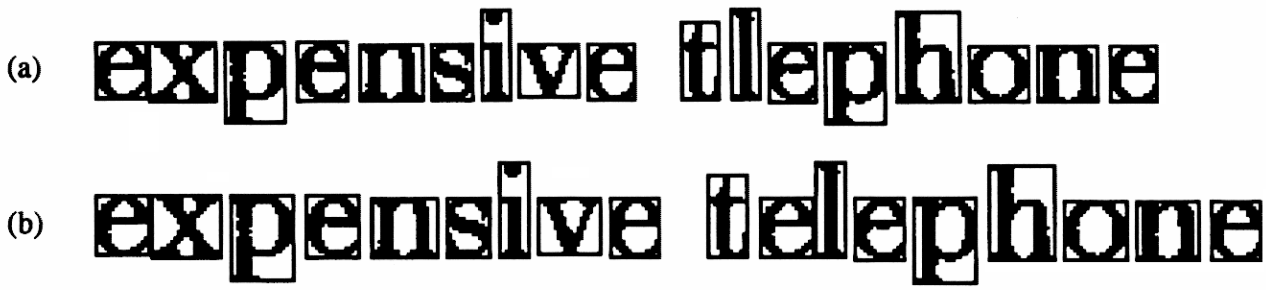
The command teach-chars puts Image EMACS into a special mode (called "teach emacs") for interactively binding keys to character images drawn from an existing text document. When the user types a key in teach mode Image EMACS binds a copy of the character under the cursor to the key and advances the cursor. After exiting teach mode, any of the "taught" character images can be inserted by typing the corresponding key.

The command key-bindings-to-buffer creates a buffer containing a visible image encoding of the current key bindings that can be edited and saved just like any other image. The command key-bindings-from-buffer causes the current buffer to be interpreted as such a representation and establishes the indicated key bindings.

Image EMACS implements search by comparing a sequence of reference character images (the search string) to successive character images in the buffer until the match score exceeds a preset threshold. The search string can be specified by typing, in which case it is constructed from the character images that are bound to the keys. Alternatively, the search string can be specified using the mouse, either by dragging the mouse across a sequence of contiguous characters or by "hunt and peck."

The search string match score at a particular buffer position is the minimum character match score between

**Figure 3.** Horizontal character positioning. A copy of the first "e" in "expensive" is inserted after the "t" in "tlephone," using copy-select-char: (a) Before; (b) After. The spaces after the "t" and "e" are preserved in "telephone."



the search characters and the corresponding buffer characters. Character matching is computed by normalized binary cross-correlation [4], that is, by counting the bits in the logical *and* of the reference and buffer characters and dividing by the geometric mean of the bit counts of the reference and buffer characters. A match is declared if the cross-correlation, which is always in the range [0, 1], exceeds a threshold.

The setting of the match score threshold affects the relative numbers of incorrect positive and negative match decisions. Experience with Image EMACS suggests that false matches are preferred to missed matches, because false hits simply increase the number of times the search for a particular string must be repeated to reach the desired instance, whereas missing a match might cause the overall search to fail. For any given setting of the threshold, match performance increases with the length of the search string. Correlation-based matching is particularly effective when the search and buffer character images are drawn from a stored font, so that scanner noise and shape variation are absent. In that case, the match threshold can be set to 1.0 (i.e., an exact image match) with the result that no false hits occur.

## Typography in Image EMACS

Image EMACS is currently built with the assumption that all text occurs in a single column of horizontally oriented lines, without any embedded multiline figures, underlining, or vertical rules. Text is assumed to be set in an ordinary "book" face without decorative features such as "drop caps." As noted previously, consecutive lines must be separated by at least one row of blank pixels.

The text column is assumed to occupy a single page defined by Cartesian pixel coordinates in which X increases to the right and Y increases downward. When an image file is read into a buffer, the top and left edges of the file image are taken to fall on the X and Y axes of the page image plane, respectively. The image plane is conceptually infinite in all directions, although there are no commands that can cause text to move to the left of the Y axis and no way to begin a line above the X axis.

The most basic typographic facilities in Image EMACS are concerned with the horizontal and vertical positioning of characters within a line of text. An early version of Image EMACS [8] adopted a simple approach to horizontal letterspacing in which the space after each character is preserved as changes are made to the text image. (More precisely, the distance between adjacent character bounding boxes is preserved.) This is illustrated in Figure 3, which adds character bounding boxes to the example of Figure 2 to show the intercharacter spacing. The space between the 't' and 'e' in 'telephone' in Figure 3(b) is equal to the space between the 't' and 'l' in 'tlephone' in Figure 3(a). Similarly, the space between the 'e' and 'l' in 'telephone' is equal to the space between the 'e' and 'x' in 'expensive.'

The letterspacing strategy illustrated in Figure 3 allows the amount of space that precedes or follows a character to depend (implicitly) on the identity, style, and size of the character. As a result, it tends to produce more satisfactory results than simply using uniform spacing. However, problems frequently arise in the case of characters with negative sidebearings (e.g., 'j') or when pairwise kerning spacing adjustments have been used. The problem of negative sidebearings is illustrated in Figure 4(b), where the spaces after the 'b,' 'e,' and 'j' are the same as in Figure 4(a).

More recent versions of Image EMACS have used an improved letterspacing algorithm in which *font metrics* (left and right sidebearings, descent below baseline) are estimated for each character [7]. The estimated font metrics are then used to position the characters using the common *sidebearing model* of character shape and positioning [1]. Figure 4(c) shows an example of character spacing using estimated font metrics.

Vertical positioning of an inserted character is accomplished by aligning the baseline of the character with the baseline of the surrounding text. The required baselines are estimated as part of the font metric estimation procedure.

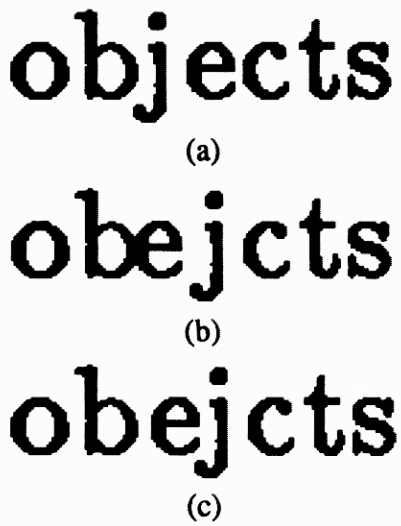



**Figure 4.** Comparison of two letterspacing algorithms for text image editing: (a) Original image; (b) The space after each character is preserved during editing; (c) Intercharacter space computed using estimated font metrics.

The accuracy of the font metric estimation algorithm is a function of the size and statistical composition of the text in the image. While the algorithm is generally reliable, unacceptable results sometimes occur, particularly for images containing relatively few characters. Because correct character alignment is critical for visual quality, several commands are available for manually adjusting character position.

A number of commands are provided for adjusting the interword spacing in a line and the alignment of a line relative to the line above it. For example, the command fill-line adjusts all of the interword spaces in a line until the right edge of the line is aligned with the right edge of the previous line, holding the left edge of the line fixed. This command is typically used to reestablish right-justification for a line of text that has changed in length—an application of this command will be described. Similarly, the command indent-line slides a line left or right to align its left

edge with the left edge of the previous line, and the command **centerline** centers the line relative to the previous line. These two commands move the line rigidly, without changing the interword spacing.

### Multilingual Text Editing

Documents containing material in writing systems other than the Roman alphabet (e.g., Japanese or Russian), as well as those using specialized graphical notations (e.g., mathematics or icons for chess pieces) present a challenge to document processing systems founded on the philosophy of format conversion. Each new type of embedded material increases the difficulty of the recognition task and the complexity of the resulting representation. Moreover, format conversion is typically an all-or-nothing proposition, so that inability to transcribe some material (e.g., if it uses a writing system that the recognizer cannot handle) implies the material cannot be edited and usually means it cannot even be reproduced.

Image EMACS, by contrast, is based on a principle of minimal (or partial) interpretation in which a text image is interpreted only to the extent necessary to support a specific set of text editing operations. As previously discussed, Image EMACS analyzes a text image into a vertical sequence of "lines," where each line is a horizontal sequence of "characters." A line is defined as a horizontal strip of image that is bounded above and below by blank rows. Similarly, a character is defined as a collection of image-connected components assembled according to simple grouping rules. Thus, the terms "line" and "character," while suggestive of rich semantics, actually correspond to relatively weak assumptions about the geometric structure of typeset English.

The geometric assumptions underlying Image EMACS are satisfied by the written forms of many languages besides English, as well as by notational systems other than typeset text. Thus, although Image EMACS was originally developed for editing images of typeset English, it is applicable to a much larger class of documents, including those containing more than one writing system. Moreover, if the assumptions underlying Image EMACS fail for some portion of an image, other regions can still be edited. Furthermore, even those regions for which the analysis fails can be reproduced with full fidelity. The analysis performed by Image EMACS is always invertible, so that *analysis "success" or "failure" refers not to the completeness of the resulting representation, but rather to the degree to which the structure of the representation matches the user's conceptual model of the material.*

Figure 5(a) shows examples of Chinese and Vietnamese text containing embedded English numerals. Figure 5(b) shows the result of editing the embedded text to replace occurrences of '586' with '510' and '330' with '310.' Because the geometric layout of the text satisfies the Image EMACS assumptions about line structure, the image was correctly parsed into a sequence of four lines. The individual connected components of most of the Vietnamese and Chinese characters were correctly grouped. A few of the Chinese characters comprise component subgroups that are separated by an all-white vertical column; these were erroneously broken into two separate characters. Most importantly, the English numerals were correctly identified as individual characters, which enabled successful editing. The appearance of the remainder of the text was preserved, even though the analysis was partly in error.

**Table 1.** Examples of Image EMACS linear text editing commands.

| Cursor Movement Commands |
|---|
| move-to-end-of-line |
| move-to-next-char |
| move-to-next-line |
| move-to-next-word |
| move-to-end-of-buffer |
| move-to-mouse |
| **Insertion and Deletion Commands** |
| delete-char |
| cut-line |
| cut-region |
| cut-word |
| paste-cut-text |
| copy-mouse-char |
| **Key Binding Commands** |
| teach-chars |
| key-bindings-from-font |
| key-bindings-to-buffer |
| key-bindings-from-buffer |
| search-for-string |

### Text Reformatting

The appearance of a given page of text results from combining a stream of textual material (the "content" of the document) with the formatting information that determines the positioning and layout of the material. Image EMACS can be used to modify the layout structure of scanned text without disturbing the content or font of the text. This type of reformatting can be used, for example, to collect the text of an article that hopscotches across several pages of a magazine onto a single page. Compared with a conventional document recognition approach to this problem, reformatting at the image level is guaranteed to introduce no content errors and will preserve more aspects of the "look" of the original material, such as unique fonts.

Figure 6 shows the result of using Image EMACS to reformat the text in Figure 1 into a slightly wider column. This operation involves performing enough image analysis to represent the image as a sequence of lines of word-sized units, but does not necessarily require complete segmentation into characters. One exception is that hyphens at the ends of lines must be isolated so that they can be removed if necessary. In this example, the hyphens were manually removed; a more sophisticated system might perform enough character recognition to detect hyphenation automatically.

### Differential Text Comparison

Differential text comparison is a useful tool for identifying editing changes in scenarios such as exchanging document drafts with a coauthor or comparing successive releases of source code or documentation. The objective of text image comparison is to summarize the changes between two separately scanned versions of a text document. Figure 7 shows a

(a) 想知道有關415/586和213/330號頭電話更改資料的中文翻譯，請打免費電話811-6888。

Để biết thêm tin tức về sự thay đổi số khu vực 415/586 và 213/330 bằng tiếng Việt xin gọi số điện thoại miễn phí 811-5315.

(b) 想知道有關415/510和213/310號頭電話更改資料的中文翻譯，請打免費電話811-6888。

Để biết thêm tin tức về sự thay đổi số khu vực 415/510 và 213/310 bằng tiếng Việt xin gọi số điện thoại miễn phí 811-5315.

**Figure 5.** Editing English text embedded in a multilingual document: (a) Chinese and Vietnamese texts containing English numerals; (b) Result of replacing "586" with "510" and "330" with "310".

simple example of image-based text comparison implemented in Image EMACS.

Figure 7(a) shows a portion of the original ("before") buffer and Figure 7(b) shows the corresponding portion of the final ("after") buffer. Figure 7(c) shows a copy of the original buffer annotated with proofreading marks that summarize how it can be transformed into the final buffer through a sequence of character insertions and deletions. A deleted character is displayed with a horizontal line drawn through it. An inserted character is written in the left margin with a caret below the line to indicate the insertion point.

Buffer comparison is performed using a standard dynamic programming method [12] of string comparison with bitmap cross-correlation to match individual characters. The Image EMACS implementation of buffer comparison is character oriented and treats all whitespace characters (i.e., CR and Space) as equivalent. As a result, text comparison is based solely on reading order and is insensitive to the location of line breaks. In this respect, it differs from line-oriented file comparisons such as the Unix diff utility. However, there is nothing essential about our particular design decision—it was intended primarily to make comparison insensitive to text reformatting (unless re-

**Figure 6.** An example of image-based text reformatting. This is the same text as Figure 1 reformatted to a slightly wider column. The extra hyphens at the ends of the lines of the original text were removed manually.

XEROX operates a private long-distance telephone system called Intelnet. The netword consists of 12 interconnected AT&T telephone switching centres with circuits to every major location in the United States, Europe, Canada and Mexico. Intelnet, when properly used, can be an inportant productivity tool, providing faster, more convenient and less expensive tlephone service than the public long-distance networks.

formatting causes changes in hyphenation).

As the text comparison example illustrates, string comparison by image correlation can form the basis for implementing a variety of text image file utilities that are analogous to conventional text file operations. Beside `diff`, other examples from Unix include `awk`, `cat`, `colrm`, `expand`, and `grep`.

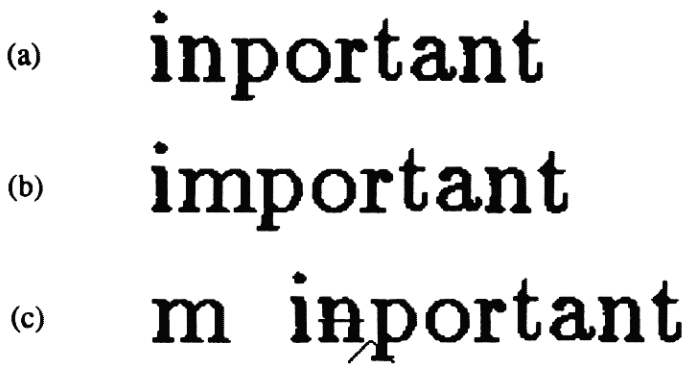
### Preprocessing for Character Recognition

The state of the art in character recognition supports the transcription of simple text with accuracy sufficient for many applications. Less adequate are currently available techniques for recognizing more complex graphical languages such as mathematical notation and for automatically reconstructing formatting and layout information from scanned documents. Often, formatting commands and mathematics are added to a document manually, as a postprocessing operation after the basic text has been recognized. The process of adding markup can be tedious and error-prone, in part because it involves frequent shifts of attention between the original document image and the emerging electronic representation. The need to shift attention can be reduced significantly by adding markup to the scanned document image *before* character recognition. The objective is to preprocess the image into a form in which the formatting and layout information is explicitly present as text that can be directly transcribed by the character recognizer. The advantages of the preprocessing approach are: It plays to the strengths of existing OCR systems, rather than their weaknesses; and it allows the user's attention to remain fixed on a single document that displays both the desired output and the user-generated formatting commands, following the "what you see is what you get" principle of interactive system design.

Figure 8(a) shows a buffer containing a small portion of a scanned document originally formatted using LATEX. Figure 8(b) shows the buffer after it has been edited into (the image of) LATEX source code for creating the image in Figure 8(a). Most of the editing involves inserting appropriate commands for setting the font and entering or leaving math mode. However, note that the math symbol '↦' has been replaced by the LATEX code fragment '`\mapsto`', since it was known that the available character recognizer could not handle that symbol. Finally, Figure 8(c) shows the result of transcribing the annotated buffer with an omni-font character recognizer.

The image-based preprocessing approach was recently employed by one of the authors to transcribe a 50-page technical paper into LATEX in preparation for revision. (The paper was several years old and the original electronic document files were no longer available.) The transcription process took about four days, including time for implementing a number of specialized Image EMACS commands for automating certain operations. The printed document was high-quality laser printer output and it was carefully scanned to minimize defects such as skew and broken or touching characters. As a result, the character recognition process was essentially error-free and nearly all of the editing time was spent adding markup to the image before recognition. The transcription was completed in less time and with less effort than the authors expected, based on their previous experiences in document transcription, although no systematic comparison was made.

**Figure 7.** Finding the editing changes between two buffers: (a) Original buffer; (b) Final buffer; (c) A copy of the original buffer, with automatically generated proofreader's marks showing how to edit the original buffer to form the final buffer.




### Related Work

Image EMACS appears to be the first system explicitly designed to support interactive text editing of scanned document images. There are several systems for bitmap manipulation, however, that embody some of the features of Image EMACS.

The system most similar to Image EMACS in design philosophy is Scott Kim's Viewpoint [6], a bitmap editor for text and graphics that was the subject of Kim's Ph.D. dissertation. Viewpoint was conceived as an exploration of the interplay between computers and graphic arts and the way in which computers might support visual thinking. A basic feature of Viewpoint, shared with Image EMACS, is that the fundamental data structures are images rather than structured symbolic descriptions. Viewpoint applies this principle more broadly than Image EMACS; in Viewpoint, every aspect of the editor's state is represented explicitly in the screen image and can be manipulated by editing the screen bitmap. For example, an image of the keyboard appears at the bottom of the screen; by editing the image inside the picture of a key, the user changes the appearance of the character that appears when that key is pressed.

The Viewpoint screen is organized as a fixed array of 10-pixel × 10-pixel cells. This constraint greatly simplifies image parsing and allows straightforward implementation of some interesting behaviors. For ex-

that case $(d, r)$ is called a *sample* of $x$. If $x : D \mapsto R$

(a)

that case $(d, r)$ is called a {\em *sample*} of $x$.

If $x : \script{D} \mapsto \script{R}$

(b)

```
that case $(d, r)$ is called a {\em sample} of $x$.
If $x : \script{D} \mapsto \script{R}$
```

(c)

**Figure 8.** Image editing to prepare a document for character recognition: (a) Original image; (b) Image after insertion of LATEX commands; (c) Result of character recognition of annotated image.

ample, Viewpoint embodies a novel context-sensitive definition of the beginning of a line. When a CR is typed, the cursor moves down to the next row of cells and then to the left until it encounters a cell containing an image not currently in any of the keyboard cells (and therefore, by hypothesis, not a "character"). However, the cell array screen model effectively limits text to fixed-pitch fonts and restricts the class of images for which Viewpoint is useful. In particular, Viewpoint is intended only to edit images that have been created using Viewpoint and is not designed to handle scanned documents.

System Zero [9] combines typical bitmap editing functions with facilities for imposing simple typographical constraints, such as linear figure alignment and figure spacing adjustment when a figure is inserted or deleted from a line. System Zero was conceived primarily to illustrate broad theoretical issues in figural theory. Compared with Image EMACs, its support for specific text editing operations is undeveloped.

Finally, Suenaga and Nagura [15] describe a non-interactive editor that uses handwritten marks, similar to proofreader's marks, to perform cut-and-paste rearrangement of blocks of text and graphics on a scanned page.

## Discussion

The notion of image-based document processing exemplified by Image EMACS meshes well with the growing popularity of fax as a means of transmitting documents. Despite the potential technical superiority of electronic mail over fax, fax is currently used far more widely than electronic mail and the gap is still growing [3, 10, 11, 16]. A major reason for the popularity of fax is that any document image can be transmitted, regardless of formatting or content type, whereas it can be argued that only raw text can be reliably sent between electronic mail users. Because there is no universally accepted interchange standard for compound documents, structured document representations that promise great power are frequently idiosyncratic to one particular document processing system. Furthermore, the obvious approach to solving this problem—the introduction of standards—has been hampered by organizational difficulties and marketplace inertia.

Experience with Image EMACS shows that a surprising range of text editing operations can be implemented without character recognition or sophisticated image analysis. This observation, combined with the emergence of high-speed telecommunication networks supporting facsimile transmission, suggests that image-based—or more broadly, appearance-based—document processing could provide a natural and smoothly evolving path from paper-based work flows toward the ubiquitous use and interchange of electronic documents. Perhaps the primary role of structured representations will be not for interchange, but rather for facilitating the private goals of individual users. In this model, images will serve as the primary representation for communicating across system boundaries, while structured representations will be generated locally as needed to carry out operations requested by a user.

From our experience with Image EMACS we have identified several principles for the design of scanned document processing systems:

***Fidelity.*** A scanned document processing system should preserve distinctive visual appearance. Some visual features, such as fonts and logos, are the results of significant effort in graphic design. To lose or remove them without explicit request from the user lowers the quality of the document represented by the system.

*Local analysis vs. long-distance communication.* A distinction should be drawn between representations intended to support local manipulation of a document image and representations used to communicate the document between systems. Most current systems founded on the principle of format conversion confuse these two uses. Adhering to this distinction provides the system designer flexibility in choosing representations that maximize the effectiveness for the two different tasks. In particular, by keeping these tasks separate we can ensure that analysis errors do not corrupt what is communicated.

*Representation for analysis.* Given the current state-of-the-art in document recognition, each attempt to interpret the image introduces the possibility of

error. Consequently, it seems prudent to interpret the image only to the extent demanded by the particular task, in order to minimize unnecessary error. Representations for analysis should be designed to reflect a style of processing that is demand-driven, incremental, and local to the task. The primary role of structured representations might not be for storage or interchange, but rather for facilitating the private goals of users. This viewpoint also suggests that analysis representations are subordinate to the (generic) document processing operations, which apply to both scanned and structured document representations.

***Representation for communication.*** Interchange is the only operation that *requires* agreement among different sites about the representation of a document. Since interpretations are in general not sharable across system boundaries because of differences in goals, the representation chosen for sharing can reflect this focus on interoperability without considering other processing operations. A representation that imposes no interpretation is often the most appropriate one for the purpose of sharing. Document images are one such representation. The existence of high-speed telecommunication networks for signal and image transmission supports this choice.

***Longevity.*** Because the languages used to express symbolic descriptions of documents are replaced over time, they do not represent a stable base in which to encode documents for storage and subsequent retrieval. A more plausible scenario for constructing a large library of electronic documents would rely on document images for the base representation, which would be augmented by an evolving set of image-based document processing commands, including those adapted from traditional document recognition research. ◘

**References**


1. Adobe Systems Inc. *PostScript Language Reference Manual*, 2d. ed. Addison-Wesley, Reading, Mass., 1990.
2. Baird, H., Bunke, H., and Yamamoto, K., Eds. *Structured Document Image Analysis*. Springer-Verlag, Heidelberg, Germany, 1992.
3. Borenstein, N. Why do people prefer fax to email? In *Preliminary Proceedings of IFIP WG 6.5 International Symposium on Message Handling Systems and Application Layer Communication Protocols*. Zurich, Switzerland, October 1990.
4. Duda, R., and Hart, P. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.
5. Horn, B. *Robot Vision*. MIT Press, Cambridge, Mass., 1986.
6. Kim, S. *Viewpoint: Towards a Computer for Visual Thinkers*. Ph.D. dissertation, Computers and Graphic Design, Stanford University, 1987.
7. Kopec, G. *Least-Squares Font Metric Estimation from Images. IEEE Trans. Image Process. 2*, 4 (Oct. 1993), 510–519.
8. Kopec, G., and Bagley, S. Editing images of text. In *EP90*, R., Furuta, Ed. Cambridge University Press, Cambridge, U.K. 1990.
9. Levy, D. *On the design of tailorable, figural editors*. SSL Report P89-00018, Xerox PARC, Palo Alto, Calif., January 1987.
10. Markoff, J. Marrying the PC and fax machine. *New York Times*, (May 2, 1990), p. C8.
11. McCarthy, J. Networks considered harmful for electronic mail. *Commun. ACM 32*, 12 (Dec. 1989), 1389–1390.
12. Miller, W., and Myers, E. A file comparison program. *Software—Practice and Experience 15*, 11 (Nov. 1985), 1025–1040.
13. Nagy G. Towards a structured-document-image utility. In *Proceedings of International Association for Pattern Recognition Workshop on Syntactic and Structural Pattern Recognition*, (Murray Hill, N.J., June 1990).
14. Stallman, R. *GNU Emacs Manual*. Free Software Foundation, Cambridge, Mass., 1986.
15. Suenaga, Y., and Nagura, N. A facsimile-based manuscript layout and editing system by auxiliary mark recognition. In *Proceedings of the IEEE Fifth International Conference on Pattern Recognition*, 1980.
16. Wilkes, M. Networks, email, and fax. *Commun. ACM 33*, 6 (June 1990), 631–633.



**About the Authors:**

**STEVEN C. BAGLEY** is a former member of the research staff in the Systems and Practices Laboratory of the Xerox Palo Alto Research Center. Current research interests include premedical studies. **Author's Present Address:** 25 Stowe Lane, Menlo Park, CA 94025; email: bagley@camis.stanford.edu

**GARY E. KOPEC** is a member of the research staff in the Information Sciences and Technologies Laboratory of the Xerox Palo Alto Research Center. Current research interests include document image analysis, pattern recognition, and signal processing. **Author's Present Address:** Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA 94304; email: kopec@parc.xerox.com