Binary-image-manipulation algorithms in the Image View Facility

by K. L. Anderson
F. C. Mintzer
G. Goertzel
J. L. Mitchell
K. S. Pennington
W. B. Pennebaker

Most current implementations of electronic mail deal primarily with coded information. A scanned-document-handling system that could scan a document, distribute it, display it on terminals, and print it on host-attached printers would offer a similar convenience for documents in hard-copy rather than coded form. For such a system to be practical, fast software is needed for a number of image-manipulation functions. The required functions are compression, to reduce the size of the data files; decompression, to reconstruct the scanned document; scaling, to match the resolution of the scan to the resolution of the display or printer; and rotation, to reorient documents scanned sideways or upside down. This paper describes a collection of algorithms underlying fast software for manipulating binary images that is used in the Image View Facility, a System/370-based software package that permits the display and printing of binary images at various resolutions.

Introduction

Commercial replacement of paper documents with electronically stored information has focused primarily on coded data. The class of documents which can conveniently be represented in coded form is limited; such items as letters (containing letterheads and signatures), drawings, and information captured via scanners are more readily and accurately handled in noncoded (e.g., bitmap) form. The use of such data introduces problems of data volume, display, and printing which generally do not arise with coded information.

The Image View Facility (IVF) program offering [1] is a partial solution to the above problems. IVF is a System/370-based software package which allows bitmap images to be displayed or printed, permitting convenient access to electronically stored noncoded information. Its purpose is to facilitate the use of noncoded information in office and commercial environments. It deals particularly with the processing of medium-resolution binary (black/white) images on commercially available IBM hardware. IVF was developed jointly by IBM teams in the United Kingdom and at the T. J. Watson Research Center in Yorktown Heights, New York; Peter Somerville coordinated the project. The first release was announced in early 1984, and a second release followed a year later.

IVF combines a convenient and understandable user interface for specification of basic image-manipulation tasks with a set of fast processing kernels to perform those tasks.

The user-interface considerations encountered by the UK developers are discussed in Reference [2]. This paper provides a brief description of some of the underlying image-manipulation algorithms developed at Yorktown.

Speed of operation is a key feature of IVF. It is undesirable for a user to have to wait more than a few seconds for an image to be displayed, and it is advantageous to be able to have several users running IVF simultaneously without overloading the host system. The attention paid to creating fast code mandated the development of computationally simple algorithms; hence, many of the IVF algorithms can be easily recoded to execute in "reasonable" amounts of time on smaller processors. This has allowed us to carry them over to systems other than IVF with a minimum of effort.

Table 1 lists some of the I/O devices supported by IVF. A variety of image formats are supported. These include raster bitmaps, images compressed using the Scanmaster I (IBM 8815) [3] compression algorithm, and Composed Document Printing Facility (CDPF) [4] and Print Services Facility (PSF) [5] page segments and list files. Images may be viewed on the various displays and reformatted (with appropriate scaling) for printing on any of the supported printers.

The algorithms described in this paper fall into three categories: data compression, scale change, and rotation. In order to use image data received from the Scanmaster, a facsimile device (used as a scanner and printer) supported by IVF, it is necessary to decompress the data stream which is created by a compressor implemented in the Scanmaster hardware. To print an image on the Scanmaster, the image must be encoded to create the correct compressed data stream. Since various printers have differing resolutions and the capabilities of display screens vary, some scaling of images is often required to take an image from one device, and display or print it on another. Finally, since documents are not always scanned right side up, some rotation capability is desirable to display images at the correct orientation. Although software for performing these functions has existed for some years, there are few detailed descriptions of the underlying algorithms in the literature. Most of the image-manipulation algorithms described were devised for hardware systems detailed in patent applications; generally these methods are not appropriate for efficient software implementation.

In [6], Takao describes in some detail an earlier image-editing system developed at the IBM Tokyo Scientific Center (now the Tokyo Research Laboratory). This system, called IEDIT, stores images in a compressed form and manipulates them in the same form, thus eliminating the need for conversion of image data from one format to another (e.g., between compressed and raster form, as in IVF). However, the representation used by IEDIT is not as compact as that achieved by other known compression methods; it was designed to trade off compression for ease of manipulation. The development of fast compression/decompression software such as that used in IVF has made the conversion processes less of a bottleneck. Furthermore, as compression methods improve, the compression software in IVF can be augmented or replaced by better compressors without disturbing the operation of the rest of the system. Since image data must generally be converted to raster form for display or printing, little is lost by converting earlier in the process rather than later. We believe that our approach of storing the image data in maximally compressed form and converting the data to the most convenient form for manipulation allows us to take advantage of the best compression techniques available, while permitting superior speed of operation.

To illustrate the uses of some of the algorithms described below, consider the problem of formatting a Scanmaster image for presentation on an IBM 3279 display in the various modes used by IVF, as illustrated in **Figure 1**. The image must first be decompressed to create a raster image which can be scaled. This image, represented by the large rectangle, is 1728 pels (picture elements) horizontally by 2200 pels vertically; the area of the screen used for image display, represented by the crosshatched rectangle, is only 720 pels horizontally by 360 pels vertically. If a 1:1 mapping of image pels to screen pels is used, only a small portion of an image can be shown. This is useful for examining details of an image but does not provide a convenient way to identify a document or read a memo. For these purposes, the image must be reduced in resolution so that most or all of it can be viewed simultaneously. A 12:5 reduction in both dimensions, accomplished by using a 2:1 reduction followed by a 6:5 reduction, is required to reduce the image width to 720 pels. Having the entire width on the screen simplifies the user interface because it is only necessary to scroll in one

Table 1 Some IBM image input/output products supported by IVF.

Input device	*Display dimensions* (pels)	*Scan dimensions* (pels/in.)
Scanmaster		
High-resolution	1728 × 2200	200
Low-resolution	1728 × 1100	200 × 100
Displays		
3278	720 × 512	
3279	720 × 384	
3290	960 × 751	
PC/G	720 × 512	
PC/GX	1024 × 1024	
Printers		
Scanmaster		200
3800 Model 3		240
4250		600

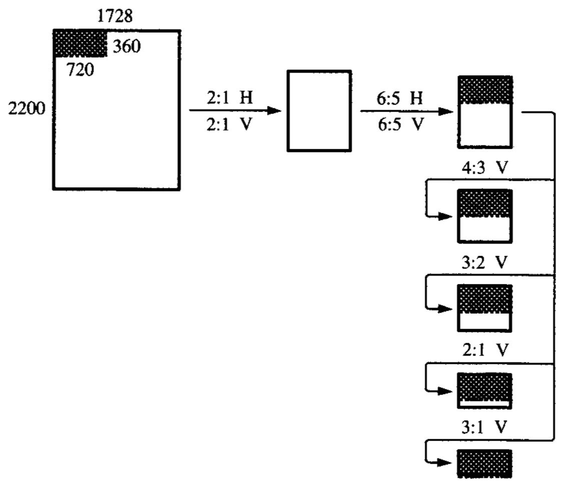


Figure 1

Display of a Scanmaster image on a 3279.

Table 2 Image View Facility kernels.

Image decompression/compression	
	Decoder
	Encoder
Scale changes	
	2:1 Reduction
	6:5 Reduction
	5:6 Enlargement
	General enlargement/reduction
	1:3 Enlargement
Rotations	
	180-Degree
	90-Degree (clockwise and counterclockwise)

dimension. Additional reductions in the vertical dimension may be performed to display varying fractions of the total image height. The 4:3 and 3:2 options display the image with approximately the correct aspect ratio (since on the 3279 the screen pels are higher than they are wide), while the 2:1 and 3:1 options are used to display most or all of the page. The vertical reductions used in the display process are performed by ORing adjacent scan lines together [2]. Presentation of the scaled image on the display is handled by the Graphical Data Display Manager (GDDM) [7]; this process is outside the scope of the present paper.

Table 2 lists the IVF kernels which have been developed at Yorktown. The decoder is used to decompress Scanmaster images for manipulation and display on other devices. The corresponding encoder takes an image in raster or "run end" form and compresses it so that it can be sent to a Scanmaster for printing. The 2:1 reduction is used to reduce images for all of the displays. The 6:5 reduction is used to convert from the 3800 Model 3 printer document resolution of 240 pel/in. to the Scanmaster resolution of 200 pel/in. and is also used to reduce images so that the full width can be displayed on the 3278, 3279, and PC/G terminals. The 5:6 enlargement does the reverse conversion. For preparing Scanmaster images to be printed on the 4250, a 1:3 enlargement is required. The general enlargement and reduction functions are used in cut-and-paste applications. A 180-degree rotation function and clockwise and counterclockwise 90-degree rotation functions are included so that the orientation of an image can be changed.

Each of these algorithms operates on images stored in one of two internal formats. Most work with images stored in raster format, in which each image pel is represented by a bit. White pels are represented by *0* bits, and black pels by *1* bits. The image bits are packed eight to a byte and stored in rows reading from left to right and from the top of the image to the bottom. Image lines always begin and end on byte boundaries. The second format is a run-end representation created by the decoder for the Scanmaster compression algorithm. In this form the horizontal position of each white-to-black and black-to-white transition in a line of the image is recorded in a vector of transition points; each image line is represented by one such vector. For typical images this representation is more compact than the raster form, but for very complex images it can consume much more space. Functions which convert data between these two formats are included in IVF.

Some assumptions concerning the types of images which are to be manipulated using IVF have been made in the process of designing the algorithms. A major assumption is that most images come from scanned documents that are binary rather than gray-scale in nature; hence, they contain text, handwriting, and line art, but not digital halftones. In addition, it is assumed that images will consist of black "information" on a white background. Thus, whenever it is necessary to choose whether to preserve a white area or a black one, it is assumed that the black area is more important. The image-reduction algorithms described produce poor results when applied to images that do not meet these criteria, and the Scanmaster compression algorithm may fail to compress or may even expand images containing large areas of halftoning.

The various image-manipulation algorithms are described below. Their execution times on various test images on a 3081KX are summarized at the end of the paper.

Image decompression and compression

Data compression is essential to make digital facsimile practical. A typical Scanmaster I image has 1728 pels per line and 2200 lines in the high-resolution mode. Each scan line can be stored in raster form as a 216-byte record with eight pels packed per byte for a total of approximately half a million bytes per high-resolution image. A low-resolution image, consisting of 1100 lines of 1728 pels each, has about a quarter of a million bytes. Low-resolution images are typically compressed by factors of 6 to 30 with the Scanmaster algorithm; for high-resolution images, the compression ratio ranges from 7 to over 40 on typical documents [8]. Thus, it is usually faster to transmit a compressed image and decode it than to send the uncompressed image. This is often true even when the communication is to a hard disk for storage on a Series/1, PC, or System/370 system.

A small portion of a facsimile image is shown in **Figure 2**. Each square represents one pel and is either black or white. Note that black and white pels tend to come in groups, or "runs." One-dimensional run-length coding takes advantage of this strong horizontal correlation by coding the number of pels between color (black/white) changes rather than storing the value of each pel. **Figure 3** shows some examples of run lengths.

Two-dimensional coding improves upon one-dimensional coding by taking advantage of the strong correlation in most images which comes from the vertical continuity of objects, strokes, or lines. Vertical-reference coding represents a run as the distance between the end of the run being encoded and the corresponding color change on the history line (preceding scanline). **Figure 4** shows examples of vertical references.

There exists a CCITT (International Telegraph and Telephone Consultative Committee) standard for both one-dimensional and two-dimensional facsimile data compression [9]. The two-dimensional standard, Modified READ, was further changed to create the Modified Modified READ (MMR) algorithm used by Scanmaster. Modified READ was designed to operate in an environment in which bit errors might be introduced in the transmission process. It therefore provides various error-checking and correction mechanisms. In a two-dimensionally coded image, an error in a single bit can propagate through many lines by causing a run end to be incorrectly identified and then used as a reference point for decoding run ends in subsequent lines. It is thus advantageous to limit the potential damage by periodically encoding an image line one-dimensionally, i.e., without reference to information on the previous line. The CCITT standard recommends that at least every second line for low-resolution images and at least every fourth line for high-resolution images be encoded one-dimensionally. In addition, each encoded image line is terminated by an end-of-line code consisting of at least eleven *0* bits followed by a *1* bit and a tag bit indicating whether the next line is encoded one- or two-dimensionally. The code words used in Modified READ are constructed so that it is impossible to have as many as eleven consecutive *0*s except at the end of a line. Thus it is always possible to recognize end-of-line codes in the code stream and break in to start decoding the first available one-dimensionally coded line following a serious error. In the Scanmaster environment, the integrity of the data is guaranteed, and so these procedures are not necessary. The Modified Modified READ algorithm uses the same vertical reference and run-length techniques as Modified READ, but requires that only the first line be coded one-dimensionally and omits most end-of-line codes, thus improving compression. The form of the MMR data

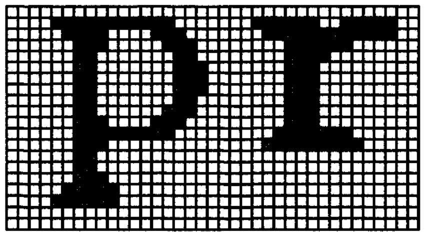


Figure 2

Raster image data.

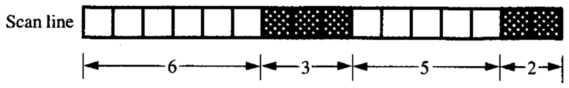


Figure 3

Run-length examples.

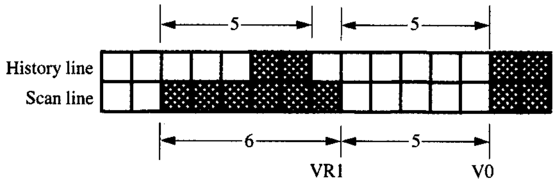


Figure 4

Vertical references.

stream describing an image is

```
1D EOL  1D data  2D EOL  2D data  2D data        . . .
1D EOL  1D EOL   1D EOL  1D EOL   1D EOL   1D EOL
```

The first end-of-line code, the 1D EOL, indicates that the first line is encoded one-dimensionally; this allows the number of pels per image line to be determined by the decoder. The next end-of-line code, the 2D EOL, marks the end of the one-dimensionally coded data describing the first line and indicates that the succeeding lines are encoded two-dimensionally. The remaining lines of the image are then encoded one by one with no intervening end-of-line codes. The series of six 1D EOL codes constitutes a return-to-control sequence, or end of page, as in the CCITT standard. Fill bits (added to the end-of-line codes to obtain a minimum transmission time per line) are not allowed. These changes typically improve the compression performance by 15 to 35 percent over that obtained using the CCITT standard for both high- and low-resolution images [8].

Scanmaster I implements the MMR algorithm in hardware; its facsimile images are available only in compressed form. A host decompressor is needed to enable the Image View Facility to display the facsimile images [2]. For editing images in compressed form, both an encoder and a decoder are needed. For IVF it is essential that encoding/decoding be performed quickly. Although conventional wisdom predicted that more than ten million instructions would be needed to decode the image of a typical business letter, the final software implementation required about one million instructions to decode such a letter (20 kilobytes in compressed form). A page of dense text (70 kilobytes) takes about three times as long.

Image scaling

Image scaling is required in IVF to increase or reduce the number of pels per row (or column) in an image in order to match the image to a printer or display device. Various systems have been proposed to enlarge or reduce images. Reduction methods range from simply discarding preselected rows and columns [10, 11] through combining clusters of pels with [12] or without [13] reference to the values of neighboring pels, to the "fast projection method" proposed by Morita et al. [14] which takes a weighted average of neighboring pels to assign values to pels in a reduced image, and the font-scaling method of Kikutani [15], which makes use of geometric reference points and lines in order to scale characters. A similar variety of enlargement systems is available, from replication of original image rows and columns [10, 13], to methods which assign values to inserted pels on the basis of logical combinations of neighboring pels [16] or detection of preselected patterns in the original data [17], to the more complex methods of Morita et al. [14] and Kikutani [15]. In addition to these procedures, which operate on raster data, there exist systems in which image information is expressed in geometric terms (e.g., the METAFONT™ system [18]) in which scaling can conveniently be done. However, such systems generally are not used to represent arbitrary image data, which IVF has been designed to handle.

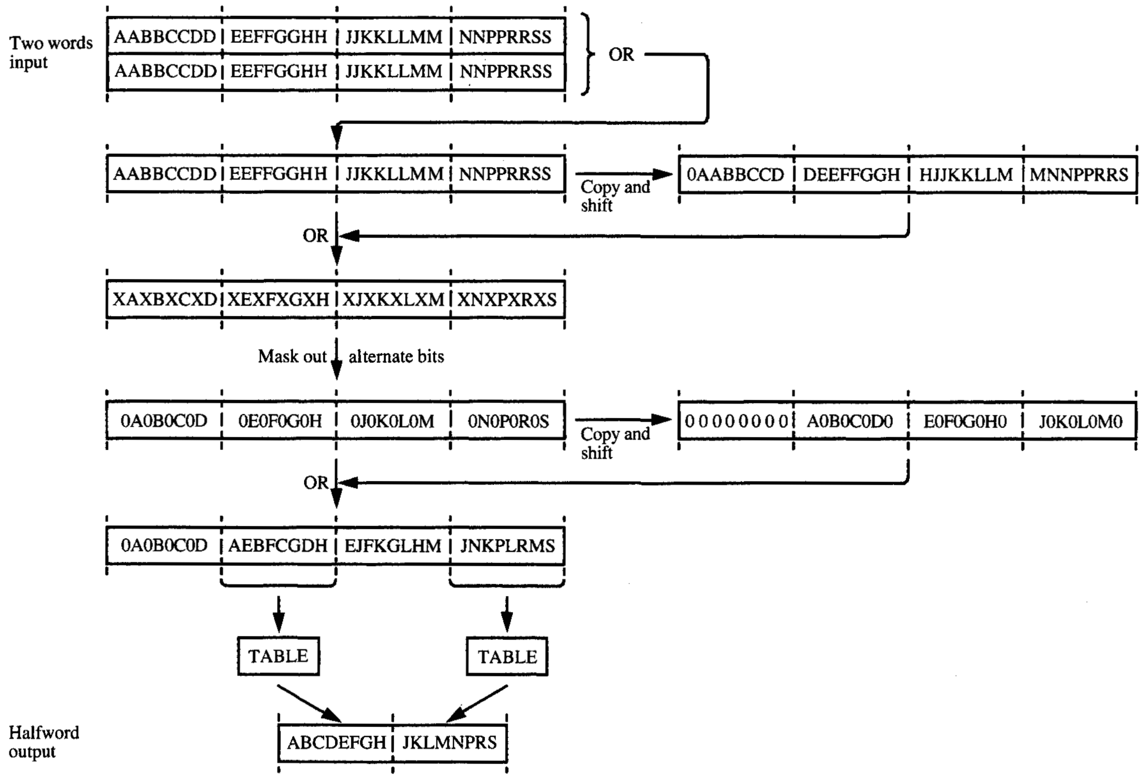
The simpler scaling methods of line-dropping or pel replication can be readily implemented in software, but the resulting image quality tends to be poor; enlarged images show staircasing on edges, while useful information may be lost during the reduction process if thin lines lie on rows or columns which are discarded. The more complex methods give better quality but tend to process images a pel at a time (most are designed for hardware implementation), resulting in unacceptable response times when the methods are implemented in software. An exception to this is Suga's method [16], which is similar in approach to the 5:6 enlargement method described later. However, Suga's method is designed for enlarging character fonts and makes certain assumptions about the bitmaps being enlarged; the consideration of a larger neighborhood of pels than Suga uses significantly improves image quality by maintaining the connectivity of thin diagonal lines.

• *2:1 Reduction*

The 2:1 reduction algorithm is used to reduce images to fit on the various displays. Since image display is a highly interactive process which usually does not result in a "permanent" image (i.e., hard copy), the speed vs. image quality trade-off favors speed more for this algorithm than it does for the other scaling algorithms to be presented. Accordingly, a very simple algorithm is used: The image to be reduced is divided into clusters of four pels, and each cluster is reduced to a single pel whose value is the logical OR of the original four pels. This method results in better image quality than simply discarding rows and columns, and executes more quickly than would a more "intelligent" procedure.

The image is processed eight bytes (two vertically adjacent words) at a time, with each eight bytes producing two bytes of output. Processing is omitted if both input words contain all *0* bits; by setting all of the bytes of an output line to zero before processing the input lines which create it, the need to store each zero result individually is eliminated. (On the System/370 it is generally faster to clear a large area such as the storage required for an image line using a single *move character* instruction than to store each zero result separately.) The test for input words which are zero allows 70 to 90 percent of typical images to be skipped over very quickly.

If the input words contain nonzero pels, they are reduced to a halfword by the procedure shown in **Figure 5**. The clusters of four pels to be ORed together are indicated by the letters representing the pels; e.g., the four pels designated "A" will be ORed together to create a single pel "A" in the output halfword. The input words are ORed together, and



Two words input
AABBCCDD | EEFFGGHH | JJKKLLMM | NNPPRRSS
AABBCCDD | EEFFGGHH | JJKKLLMM | NNPPRRSS
OR
AABBCCDD | EEFFGGHH | JJKKLLMM | NNPPRRSS
Copy and shift
0AABBCCD | DEEFFGGH | HJJKKLLM | MNNPPRRS
OR
XAXBXCXD | XEXFXGXH | XJXKXLXM | XNXPXRXS
Mask out alternate bits
0A0B0C0D | 0E0F0G0H | 0J0K0L0M | 0N0P0R0S
Copy and shift
00000000 | A0B0C0D0 | E0F0G0H0 | J0K0L0M0
OR
0A0B0C0D | AEBFCGDH | EJFKGLHM | JNKPLRMS
TABLE
TABLE
Halfword output
ABCDEFGH | JKLMNPRS

Figure 5

OR procedure for 2:1 reduction of two words.

the result is copied, shifted right one position, and ORed in. This produces a word in which the alternate bits represent the OR of the four-pel clusters. The intervening bits are not needed, so they are masked out. This result is then copied and shifted right seven positions, and the unshifted and shifted words are ORed together. This collects the bits required for the output into two bytes. The bits of each byte are placed in the correct order by using them to index a lookup table which produces for each eight-bit input an eight-bit output whose bits are the same as the index value but in a different order. The two bytes resulting from the table lookups are stored as a halfword in the output image.

- *6:5 Reduction*

In addition to its usefulness in the display process of IVF, this reduction and the matching enlargement convert between document resolutions of 200 pel/in. and 240 pel/in. One feature of these algorithms is that if a 200-pel-resolution document is enlarged using the 5:6 enlargement and then reduced back to the original resolution by the 6:5 reduction, the resulting image is identical to the original image.

The 6:5 reduction algorithm divides an image into six-by-six-pel blocks and reduces each block to a five-by-five-pel block. For convenience in processing, the original blocks are handled in units of six bytes by six rows; the corresponding output units are five bytes by five rows. If a unit contains all *0* bits, no processing is necessary, since the output area is cleared before data are moved into it. This check typically reduces the amount of data to be processed by more than half.

Figure 6 illustrates the reduction of a six-by-six-pel block. Each row is used to index a lookup table which shortens the row by one pel and simultaneously transposes it, as described below. When all of the results are summed, a five-

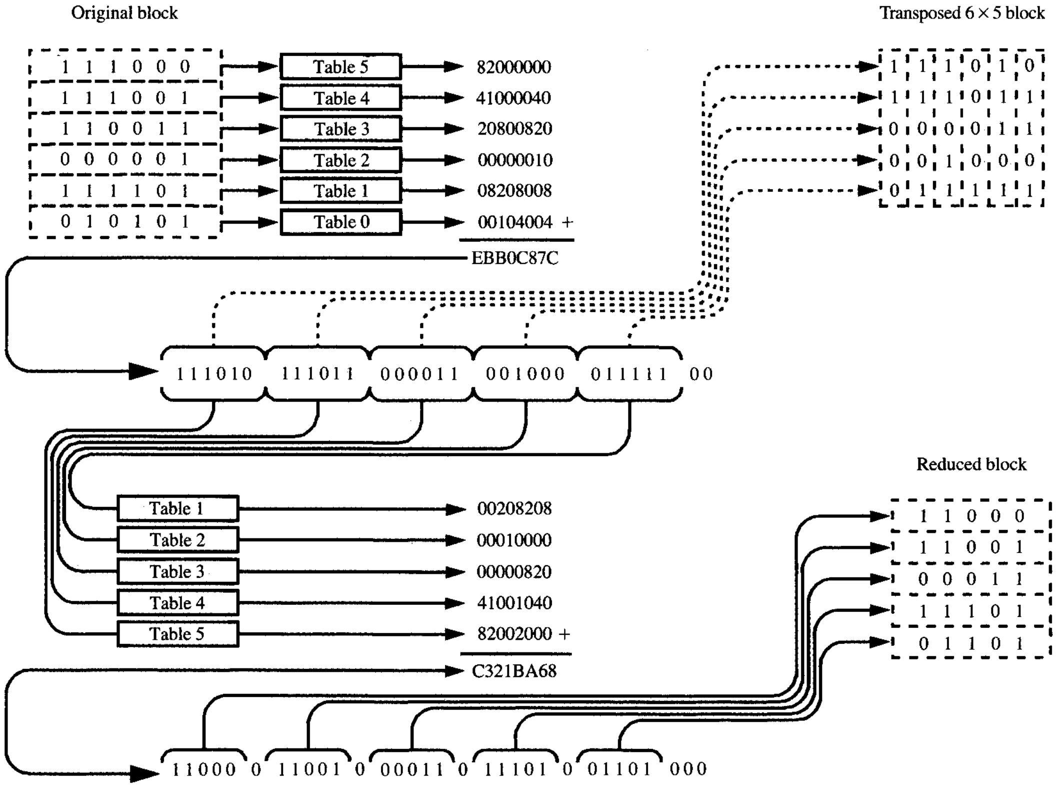


Figure 6

Reduction of a six-by-six-pel block.

by-six-pel block which approximates the transpose of the original block is produced. By repeating the process, the block is further reduced to a five-by-five-pel block transposed back to its original orientation. The objective of the row-shortening procedure is to preserve runs. If the third pel in the row can be removed without destroying a run, this is done; otherwise an attempt is made to find another run to shorten (looking for the longest run, or the run nearer the center if two runs have the same length). If the row consists of alternating white and black pels, the white pel nearest the center is removed. This process approximates simply discarding rows (or columns) except in places where a line could be destroyed by doing so. It cannot treat all runs equally; the true length of runs spanning more than one block is not considered. However, by limiting the neighborhood of pels examined to a small area, the distortion that could arise from making changes relatively far away from the line being discarded is minimized, and the computation is simplified.

A substantial amount of the potential computational effort involved in this algorithm is eliminated by constructing the lookup tables so that the row shortening and transposition are accomplished simultaneously using a single table lookup. A "straightforward" implementation of the algorithm might go through the following operations for each six-pel unit to be processed:

1. Reduce the six pels to five via a table lookup.
2. Use the resulting value to access a second table which converts five bits to four bytes containing the "exploded" original bits.
3. Shift the resulting data according to which row of the block the original six pels comprise (i.e., do not shift for the first row, shift by one bit for the second row, shift by two bits for the third row, etc.).

For example, if the bit pattern 011101 occurred as the fourth row of a six-by-six-pel block, it would be changed to 01101 by the first table lookup (the third pel is discarded), "exploded" to 00000010 00001000 00000000 10000000 (X'02080080') by the second table lookup (note that the original bit values are separated by five zero bits in the "exploded" value; two extra bits are tacked onto the end to get a computationally convenient total of 32 bits), and then shifted right by three bits to obtain the final value of 00000000 01000001 00000000 00010000 (X'00410010'). However, there is no need to do all of this work every time a six-pel unit is to be processed. The two lookup tables can be constructed, and a third table which converts a six-pel unit directly to the "exploded" reduced value can be created by applying the double table lookup to every possible six-bit input and recording the results. Additional tables can then be formed by shifting the entries in the third table by varying numbers of bits. The third table and the additional tables are used in the code which implements the 6:5 reduction; selection of the table indexed for a particular reduction/transposition operation is determined by the row of the original block that is being processed.

• *12:5 Reduction*
The 12:5 reduction required to format a Scanmaster image for display on a 3278, 3279, or PC/G is accomplished by performing a 2:1 reduction followed by a 6:5 reduction, using the algorithms described above. The 2:1 reduction is applied first because it is very fast; it reduces the data volume by 75 percent, allowing the slower 6:5 reduction to be applied to a much smaller image. Some experiments were done applying the 6:5 reduction before the 2:1, since the 6:5 reduction takes greater care to preserve the image quality. It was concluded that while the image quality is slightly diminished if the 2:1 reduction is applied first, the improvement in image-display response time obtained by using this ordering outweighs the loss of image quality.

• *5:6 Enlargement*
An image can be enlarged by inserting rows and columns and then filling them in. In the 5:6 enlargement used in the Image View Facility, the image is expanded horizontally (by adding columns) and then vertically (by adding rows). To set a bit x in an inserted line, its six nearest neighbors are examined:

$$\begin{array}{ccc} a & b & c \\ & x & \\ d & e & f \end{array}$$

If b and e are equal, x is set to their value. Otherwise, if either b or e is *1* (black) and both pels of one of the diagonals are black, x receives the value *1*; otherwise it is set to *0*. That is,

$$x = (b \,\&\, e) \mid \{[b \mid e] \,\&\, [(a \,\&\, f) \mid (c \,\&\, d)]\}.$$

Note that with the System/370's 32-bit registers, it is possible to perform this calculation for many output bits in one row simultaneously. Additional speed is obtained for typical images by observing that if either a, b, and c or d, e, and f are all *0* (white), then x must be white; zeroing of the inserted row allows storage of this result to be omitted. In practice, only one adjacent row is checked for all *0* pels; if one row is nonzero, there is a good chance that the other is also, so the overhead required to perform the second test is not justified.

To fill in bits in an inserted column, the same rule is used with the template rotated by 90 degrees. Part of the logic is incorporated into the procedure for inserting columns. Column insertion is done by dividing each image line into units of ten bits and then running each unit through a lookup table which converts a ten-bit input *mnopqrstuv* into a twelve-bit output *mnYopqrsZtuv*. By simple insertion of bit columns, the inserted bits Y and Z would receive the value *0*. However, by setting

$$Y = n \,\&\, o,$$

$$Z = s \,\&\, t,$$

the later calculation of the value of each inserted pel x is simplified to

$$x = x \mid \{[b \mid e] \,\&\, [(a \,\&\, f) \mid (c \,\&\, d)]\}.$$

Again, this calculation can be performed for several pels at the same time, masking out the result of the first part of the calculation for all of the original image bits before the final OR with the image row containing the x values.

• *General scaling*
When dealing with specific scaling factors, columns can often be inserted in or removed from an image in some regular pattern which can be programmed efficiently. When scaling by an arbitrary factor, this is not the case. For the general scaling algorithm, this problem is solved by rotating the image 90 degrees (using the 90-degree rotation algorithm described later), enlarging or reducing in the vertical dimension by the desired horizontal scaling factor, and then rotating the image back to its original orientation. The vertical scaling may be done either before or after the horizontal scaling; the size of the images to be rotated is minimized by doing the vertical scaling first if a reduction is required or last if an enlargement is specified. The scaling problem has thus been reduced to a question of how to add or remove image rows to scale in the vertical dimension. A secondary advantage of this simplification is that different scaling factors can be readily applied in the horizontal and vertical dimensions.

Our vertical-enlargement and vertical-reduction algorithms are restricted so that the size of an image may not be changed by more than a factor of two. This ensures that for image enlargement each added line will have a line of real data above and below it, and that for reduction no more than two input image rows are combined to create each output row. If greater enlargement or reduction is required, the appropriate algorithm may be applied more than once. This seems awkward when compared with more conventional methods (pel replication for enlargement, line dropping or ORing for reduction) which require only a single pass over the data for any scale factor. However, the extra work allows us to achieve better image quality than is obtained with the conventional methods. In a system where the general scaling functions are used extensively with large scaling factors, it would be worthwhile to add functions which apply similar logic to perform larger scale changes in one pass; however, since IVF typically does not make extensive use of these functions, we opted to keep the code relatively simple and compact by imposing the limitation of scaling by only a factor of two per iteration and using a simple driver to call the functions repeatedly if necessary.

Vertical enlargement

The vertical-enlargement function consists of two parts: the insertion of extra lines at appropriate intervals and the setting of the bits in the new lines. The insertion of extra lines is controlled by a counter which is incremented by a certain amount after each input line is examined. Whenever the value of the counter becomes greater than a fixed threshold, a line is inserted and the counter is decremented by another value. The threshold and the value used to decrement the counter are fixed; the increment value is calculated from the scale factor and the decrement value. This procedure allows the spacing of the inserted lines to vary so that the required scaling factor is closely approximated.

The bits in the inserted lines are set using an algorithm similar to that used in the 5:6 enlargement, except that an additional four pels participate in the calculation. The use of additional pels improves the appearance of lines with shallow slopes. For enlargement factors of 1.5 or less, the calculation is

$$\begin{array}{ccccc} a & b & c & d & e \\ & & x & & \\ f & g & h & i & j \end{array}$$

$$x = (c \mid h) \;\&\; [(a \;\&\; j) \mid (b \;\&\; i) \mid (c \;\&\; h) \mid (d \;\&\; g) \mid (e \;\&\; f)].$$

What this means is that if either of the pels above and below x is black, and if any diagonal or vertical line through x contains both black pels, then x is set to black; otherwise x is set to white. Note that since the enlargement is restricted to a factor of two or less, an inserted line always falls between two lines of original image data (with the possible exception of one line at the top or bottom of the image). As in the 5:6 enlargement, the calculation is omitted if one of the adjacent lines contains all *0* (white) pels, and otherwise multiple output bits are calculated in parallel.

A variation of the algorithm which determines the values of the pels in the inserted line is necessary if the enlargement algorithm is to be applied to double or nearly double the vertical resolution. Note that if a row of black pels is bounded above and below by white pels, the black line cannot get thicker, since an inserted bit is not made black unless both pels along a diagonal or vertical line through the inserted bit are black. This problem can be overcome by ORing one of the adjacent image lines with each inserted line. If the line below each inserted line is ORed in, the calculation of x becomes

$$x = h \mid (c \mid h) \;\&\; [(a \;\&\; j) \mid (b \;\&\; i) \mid (c \;\&\; h) \mid (d \;\&\; g) \mid (e \;\&\; f)],$$

or, equivalently,

$$x = h \mid \{c \;\&\; [(a \;\&\; j) \mid (b \;\&\; i) \mid (d \;\&\; g) \mid (e \;\&\; f)]\}.$$

This change reduces the ability of the algorithm to smooth some slanted lines, but on balance it gives a more acceptable image.

Vertical reduction

Vertical reduction of an image is accomplished by throwing away rows of image data and preserving some of the information they contain by adding it to adjacent rows. The vertical reduction operates by throwing away rows of image data after ensuring that this will not throw away any vertical black runs, i.e., black pels which have white pels both immediately above and immediately below them. This prevents such problems as the conversion of an "e" into a "c" in the case where the horizontal bar of the "e" is one pel thick and the row on which it lies happens to be discarded. If the rows before and after the row to be discarded both contain white pels at any position where the row to be discarded contains a black pel, the corresponding pel is made black in the preceding row. The selection of lines to be discarded is controlled by a process similar to that used in the general vertical enlargement to determine when to insert lines.

The vertical-enlargement and vertical-reduction algorithms are coordinated so that the reduction algorithm exactly reverses the enlargement.

- *1:3 Enlargement*

The 1:3 enlargement algorithm converts a Scanmaster image of 200 pel/in. to an image of 600 pel/in. suitable for printing on the IBM 4250 [19, 20]. It is useful to have a special-purpose enlargement algorithm to avoid the rotation of very large images which would be required in applying the general enlargement to an entire facsimile image page. This algorithm works on a few rows of the image at a time, so that not only is the rotation avoided, but it is not even necessary to keep the entire input or enlarged (output) image in storage at one time. The enlargement algorithm is designed to smooth some of the staircasing which would be produced by simply converting each pel to a three by three block of pels of the same color, and to partially compensate for the fact that the spot size of the 4250 is much larger than its addressability.

A run-end representation is used for the image to be enlarged. In this form, it is reasonably straightforward to interpolate lines in the vertical dimension. The process works on four rows of the image at a time, interpolating between the middle two. For each pair of run ends of the same color, there are two possibilities, as illustrated in **Figure 7**. In this figure the doubly-hatched areas represent black pels on the middle two original image lines; the singly-hatched areas are interpolated black pels. The run ends on the two input lines are designated B_k and C_k. If the run following the earlier run end extends beyond the later run end [e.g., $C_i < B_i < C_{i+1}$, as shown in Figure 7(a)], then the two runs define an edge of some feature in the image. In this case, an approximate interpolation is performed to determine the run ends for the new lines being created. If the run following the earlier run end does not extend beyond the later run end [e.g., $C_i < C_{i+1} \leq B_i$, as shown in Figure 7(b)], then one line contains a run "inside" the feature defined by the other run. In this case, the edge of the intruding feature is rounded. The interpolations prescribed may be modified by some additional rules which improve the quality of the resulting image. For example, white streaks less than three pels thick will be engulfed by the surrounding black pels, so they should not be created; thus, one interpolated line may be forced to have a white run extending to the end of a white run on the adjacent "original" line rather than ending at an interpolated point. Other rules attempt to preserve square corners, require black streaks in the enlarged image to be at least two pels thick, and require "four-corners" situations to be preserved.

The horizontal-enlargement procedure is designed to duplicate the effect that would be obtained by rotating the original image by 90 degrees, applying the vertical-enlargement algorithm, and then rotating the resulting image back to the original orientation. The horizontally expanded image is approximated by simply tripling each run end in the original image line, and then selectively clipping or adding single pels by modifying the run ends. The algorithm which performs this "smoothing" is based on the observation that vertical edges of the image may be "followed" over many lines. When an edge shifts by more than one pel, a square corner exists and no smoothing is required. If an edge shifts by a single pel, then smoothing is performed as necessary. For example, if an edge shifts left one pel and the previous shift in the edge was also to the left, then a linear interpolation is performed by adding one pel to the approximated run end on the first third of the lines containing the edge, leaving the middle third of the lines at

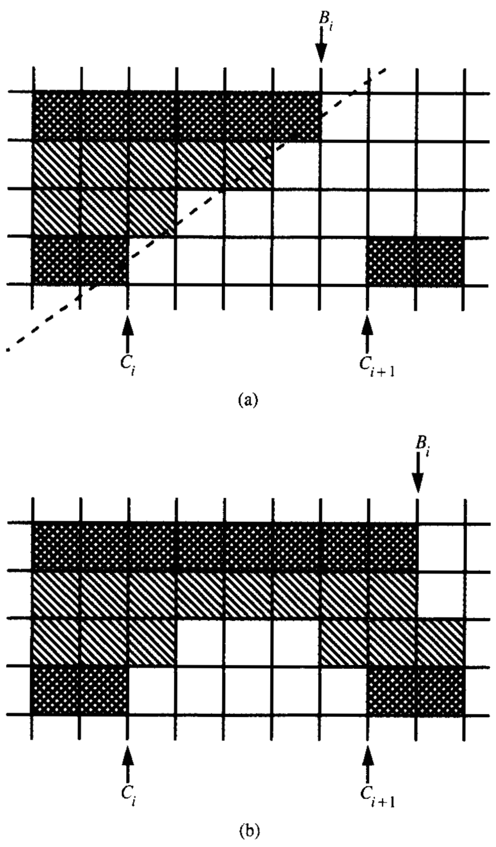


Figure 7

1:3 Enlargement algorithm: (a) Edge of feature; (b) intrusive feature.

the approximated value, and clipping one pel from the run end on the last third of the lines containing the edge. This method requires that a number of recently expanded input lines be kept in storage so that they can be modified as required. For this reason, the horizontal enlargement is done before the vertical enlargement; if a vertical edge is an inch long, it is then necessary to store only 200 lines instead of 600. The amount of storage used by the enlargement procedure is controlled by the calling function; if the storage is exhausted before the extent of some of the vertical edges has been determined, the lines are sent out through the vertical-enlargement procedure without smoothing. This may result in some staircasing. However, for typical images a storage area of 50 kilobytes is sufficient to store enough lines to prevent this problem from arising.

The output image is thus constructed in run-end representation. It can be converted to raster form or encoded (compressed) directly using the MMR encoder, which has the option of taking its input in run-end form. The fact that

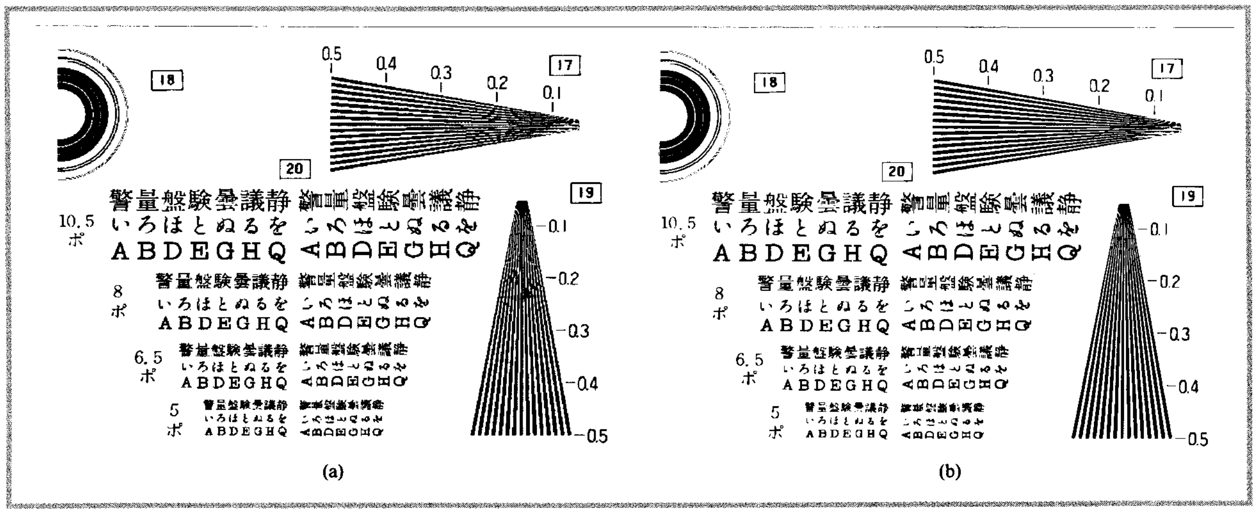


Figure 8

1:3 Enlargement (a) by pel replication; (b) with smoothing.

this algorithm works with images in run-end form can be very useful given the huge amount of raster data in a 4250 image.

Figure 8 contrasts the results of enlargement by simple pel replication with enlargement using the technique described above. Figure 8(a) shows an image section enlarged by replicating each pel to form a three-by-three-pel block, and Figure 8(b) shows the same data enlarged and smoothed using the method described earlier. Both images have been enlarged by an additional factor of two using pel replication in order to make the differences more clearly visible.

Image rotation

Rotation of images is required in a system such as IVF to reorient images which have not been scanned right side up. For this purpose, rotation by increments of 90 degrees is generally adequate. While various hardware systems are known which perform both 180- and 90-degree rotation [21–24], generally these methods are not conveniently implementable in software. In addition, the methods which physically rearrange the data do not rotate an image of arbitrary dimensions in place; images are rotated as they are read in (as from a scanner, [22]) or read out (as to a display, [24]). In a system such as IVF, it is often desirable to conserve storage by performing image-manipulation operations *in situ.*

- *180-Degree rotation*

The simplest of the rotation algorithms is the 180-degree rotation. **Figure 9** shows the effect of a 180-degree rotation on an image having six rows of four units. (In our implementation, a unit consists of four bytes.) Note that the rotation merely exchanges pairs of units and reverses the order of the bits within each unit. The 180-degree rotation algorithm in the Image View Facility rotates an image in place by positioning two pointers at the center of the image and then stepping them in opposite directions until they reach the ends of the image. At each step the units addressed by the pointers are swapped, and the bits are reversed. A substantial improvement in the speed of execution is obtained by observing that if the two units both contain all *0* pels, it is not necessary to do anything. Simple images can be rotated faster than a worst-case image by a factor of about three.

The procedure described above rotates an image in place. To rotate an image and place the output in a different area of storage, one pointer is positioned at one end of the input image and the other addresses the opposite end of the output image. The pointers are stepped in opposite directions, and at each step the bits in the input unit addressed are reversed and the results are transferred to the output-image area. This is essentially the procedure described by Ikeda [24]. The output image may be zeroed before processing begins in order to eliminate the need to store units containing only *0* pels.

Since the System/370 has 32-bit registers, our implementation of the rotation algorithm uses a unit size of four bytes. The bit reversal is done using a table lookup to reverse the order of the bits in each byte; the reversed bytes are put together in a register and then stored as a unit. Any leftover bytes are handled individually after all of the complete units have been exchanged.

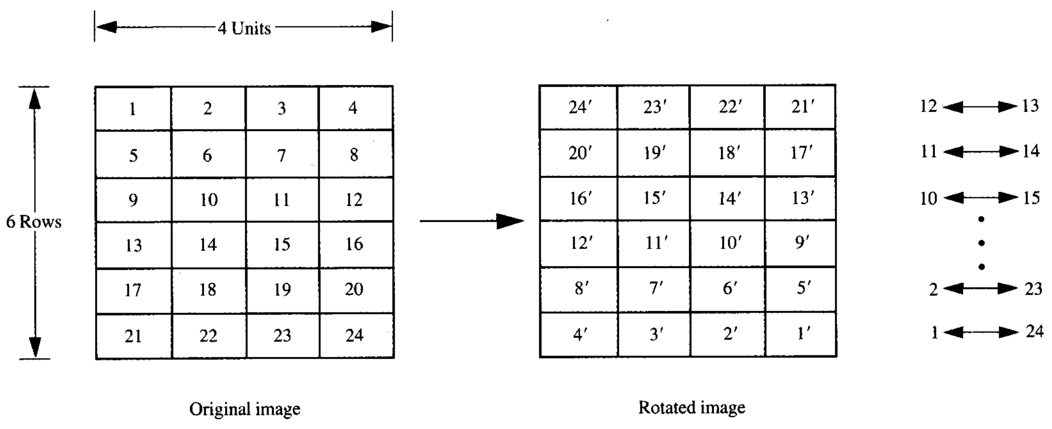


Figure 9

Unit rearrangement for 180-degree rotation.

• *90-Degree rotations*

The 90-degree rotations are similar to the 180-degree rotation in that the image is divided up into small units which are moved around and which also require some internal rearrangement. For 90-degree rotation the basic unit is the eight-by-eight-pel (one byte by eight rows) block. As in the 180-degree rotation, the internal rearrangement of pels in a block may be omitted if all of the pels are white. Even in complex text or graphics images, typically two thirds of all blocks fall into that category, and the simpler the image the more rapidly the rotation operation can be performed. The counterclockwise rotation will be described; the same procedure can be used with minor alterations in the control structure and lookup tables to perform a clockwise rotation.

Figure 10 illustrates the principle of the block rearrangement required to rotate an image in place. The image is divided into columns of some fixed width (in our implementation, 32 bytes). The leftmost column is copied to temporary storage, and the remaining data are copied up to the beginning of the image area. The cleared space is zeroed, so that no storage operations are required for all-*0* eight-by-eight-pel blocks. The image in temporary storage is then broken into eight-by-eight-pel blocks which are rotated and positioned in the cleared space to produce the last lines of the rotated image. The second column is then copied to temporary storage, the remaining original image data are moved up, and space is cleared for the next set of rows of output-image data. These rows are created by rotating the blocks of data in temporary storage and storing the rotated blocks in the appropriate places. The process is repeated until all of the columns of the original image have been copied and rotated. In practice the image data do not appear in storage exactly as shown, since storage is one-dimensional rather than two-dimensional as illustrated. However, the principle of copying part of the data, moving the remainder up to free the space required for the output, and creating the rotated image data a strip at a time still applies. This method works even if the original image is not square.

The rotation of an eight-by-eight-pel block may be done by dividing the block into 16 units of four bits, running each unit through a lookup table (which "explodes" the bits by inserting groups of seven *0* bits between the original bits and then shifts the result according to which row of the block the unit came from), and summing the results appropriately. **Figure 11** shows the calculation; note that it is simply a series of table lookups and additions.

For clockwise rotation, the block rearrangement proceeds by copying the rightmost column of the image data remaining at each step rather than the leftmost. The lookup tables used in the block-rotation process are modified, and the low-order four bits of each row of an eight-by-eight-bit block determine the last four bytes of the output block rather than the first.

Implementation and performance

The algorithms described in this paper have been implemented and packaged as CMS text decks and are available as part of IVF. The functions have no I/O; they operate on an image in storage and produce an output image in the same area of storage or in a different area. The calling function is required to set up a small number of parameters (the interface is not exactly the same for all functions);

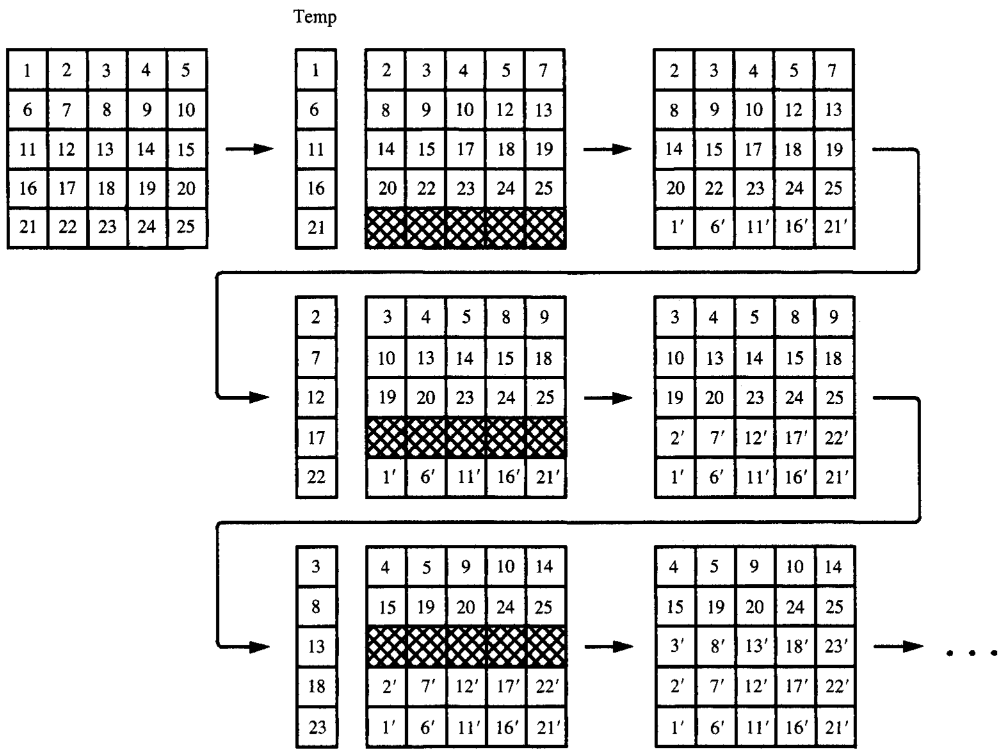


Figure 10

Block rearrangement for 90-degree counterclockwise rotation.

typically it can specify the input- and output-image areas, the size of the input image, and some information which allows the input image to be clipped out of a larger image and/or specifies that the output image is to be constructed as a subimage of a larger image.

Table 3 shows the execution times for the various functions described in this paper. The timings given are virtual CPU times expressed in milliseconds on an approximately 7.9-MIPS (million instructions per second) processor. The timings were made on a 3081KX which runs at 15.8 MIPS and has two processors, only one of which is used at any give time for a given application. Execution times are listed for operations performed *in situ*, not including I/O (with the exception of the MMR decoder and encoder, which include I/O time for the compressed data stream but not the raster image). All of the test images are 216 bytes in width; the length varies and is indicated in the table.

Conclusion

This paper has described a set of fast image-processing algorithms for performing decompression, compression, scale changes, and rotations on binary images. Techniques such as the use of table lookups and logical operations; processing words, halfwords, bytes, or run ends rather than bits; and checking for white areas in the image make it practical to perform such operations in software on currently available processors, and will make them even more attractive as processors become more powerful. In the area of image scaling, our strategy has been to have the general algorithms available, but to write separate functions to perform operations which are frequently required. Such special-purpose functions generally are simpler and execute more quickly than the general functions.

Practical image-manipulation functions contribute to the possibility of widespread use of noncoded information in the office environment. In addition to their specific uses in the

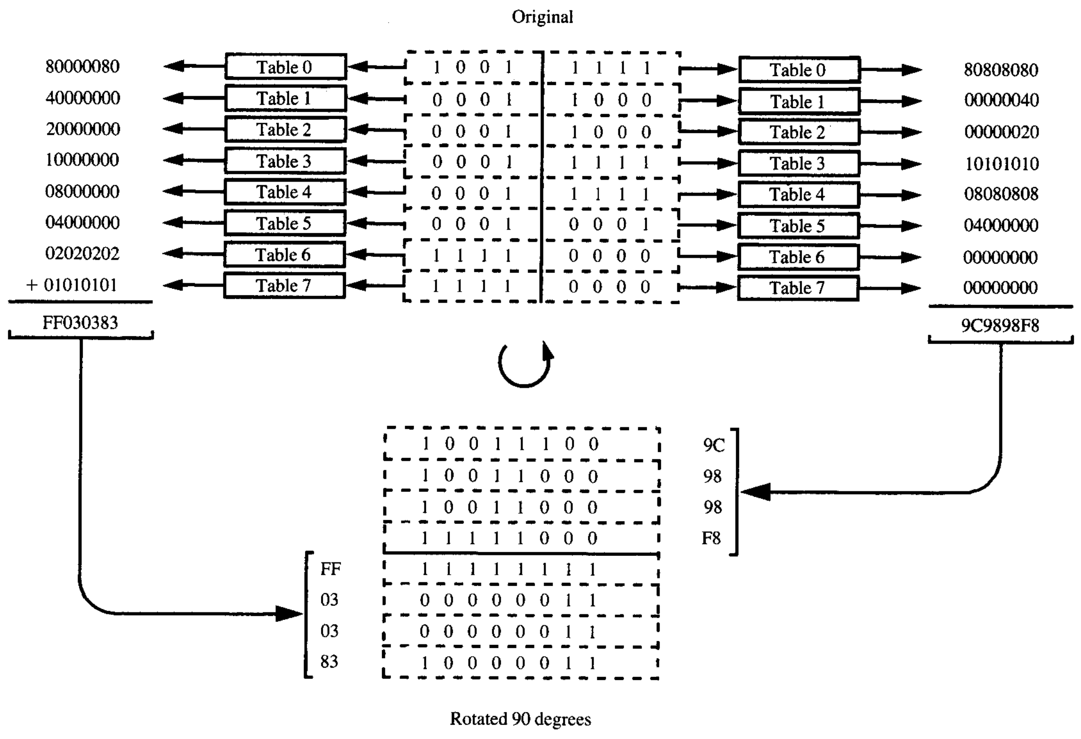


Figure 11

Eight-by-eight-bit counterclockwise rotation calculation for one block.

Table 3 Execution times for IVF subroutines. All timings are virtual CPU times on a 7.9-MIPS processor, expressed in milliseconds. All of the test images are 216 bytes in width.

Function	*Image* (No. of lines)					
	White (2128)	*Letter* (2128)	*Memo* (2163)	*Dense Text* (2128)	*Kanji* (2376)	*Checkerboard* (2128)
MMR decoder	36	117	156	344	382	5857
MMR encoder	35	119	161	354	416	6122
2:1 Reduction	21	26	30	37	43	50
6:5 Reduction	62	101	130	195	242	430
5:6 Enlargement	59	83	106	137	185	328
1:3 Enlargement/smoothing	221	378	440	928	879	8388
General vertical enlargement						
Scale factor 1.1	26	28	30	31	34	46
Scale factor 2.0	69	86	101	128	145	277
General vertical reduction						
Scale factor 1.1	25	24	25	25	25	27
Scale factor 2.0	34	35	35	35	36	44
General scaling						
Scale factor 0.5	103	133	151	197	200	281
Scale factor 0.91	191	249	276	354	380	600
Scale factor 1.1	238	299	352	453	481	796
Scale factor 2.0	488	617	701	924	976	1855
90-Degree rotations	85	115	135	180	187	320
180-Degree rotation	21	31	39	53	73	117

Image View Facility, the algorithms described in this paper have general utility for other image-manipulation applications.

Acknowledgments

The authors thank Peter Somerville for originating and coordinating the Image View Facility project, and John Dawkins and Gary Wooding for their cooperation in defining interfaces and integrating our code into the system.

References and notes

1. *Image View Facility Program Description and Operations Manual*, Order No. SB19-5919 (Program No. 5785-ECX), IBM Corporation; available through IBM branch offices.
2. P. J. Somerville, "Uses of Images in Commercial and Office Systems," *IBM Syst. J.* **23**, No. 3, 281–296 (1984).
3. *IBM Scanmaster I (Machine Type 8815) Description Manual*, Order No. GA18-2094, IBM Corporation; available through IBM branch offices.
4. *Composed Document Printing Facility (CDPF) General Information Manual*, Order No. GC33-6133, IBM Corporation; available through IBM branch offices.
5. *Print Services Facility Data Stream Reference*, Order No. SH35-0073, IBM Corporation; available through IBM branch offices.
6. Yoichi Takao, "An Approach to Image Editing and Filing," *Tokyo Scientific Center Report G318-1554*, IBM Japan, November 1981.
7. *Graphical Data Display Manager (GDDM) General Information Manual*, Order No. GC33-0100, IBM Corporation; available through IBM branch offices.
8. Joan L. Mitchell, "Facsimile Image Coding," *AFIPS Conf. Proc.* **49**, 423–426 (1980).
9. Roy Hunter and A. Harry Robinson, "International Digital Facsimile Coding Standards," *Proc. IEEE* **68**, No. 7, 854–867 (1980).
10. Robert E. Shirley, "System and Method for Generating Enlarged or Reduced Images," U.S. Patent 4,394,693, 1983.
11. I. Kitazawa, T. Uchida, and T. Ushiroda, "Reduction of Image," *IBM Tech. Disclosure Bull.* **27**, No. 5, 3019–3020 (1984).
12. Ian D. Judd, "Method and Means for Scale-Changing an Array of Boolean Coded Points," U.S. Patent 4,280,143, 1981.
13. Everett Truman Eiselen, "Apparatus for Image Manipulation," U.S. Patent 3,976,982, 1976.
14. Hideki Morita, Masatoshi Maeda, and Yasuhiko Yasuda, "A Resolution Conversion Scheme for Black-and-White Images," *IEEE Global Telecommunications Conference Record*, San Diego, CA, 1983, pp. 1255–1260.
15. M. Kikutani, "Image Scaling," *IBM Tech. Disclosure Bull.* **27**, No. 5, 2984–2986 (1984).
16. Gojiro Suga, "Dot Matrix Converter," U.S. Patent 4,090,188, 1978.
17. Setsuo Yonezawa, Tsuneta Kawakami, Tatsuo Shimada, and Yoshinori Chida, "Apparatus for Forming a Character by a Matrix Pattern of Picture Elements," U.S. Patent 4,129,860, 1978.
18. Donald E. Knuth, *T_EX and METAFONT: New Directions in Typesetting*, American Mathematical Society and Digital Press, Bedford, MA, 1979. METAFONT is a trademark of the Addison-Wesley Publishing Co., Reading, MA.
19. *IBM 4250 Printer Component Description and Programming Information*, Order No. GA33-1554, IBM Corporation; available through IBM branch offices.
20. Gerald Goertzel and Gerhard R. Thompson, "Digital Halftoning on the IBM 4250 Printer," *IBM J. Res. Develop.* **31**, No. 1, 2–15 (1987, this issue). See especially the Appendix (*The IBM 4250 Printer*), which summarizes the relevant techniques, materials, and media.
21. D. E. Gold, T. H. Morrin, and D. C. Van Voorhis, "Shift Register System for Image Orientation," *IBM Tech. Disclosure Bull.* **18**, No. 8, 2633–2639 (1976).
22. Peter J. Evans, "Image Rotation Apparatus," U.S. Patent 4,168,488, 1979.
23. Jeffrey B. Lotspiech, "Method and Apparatus for Rotating the Scan Format of Digital Images," U.S. Patent 4,271,476, 1981.
24. Yasuo Ikeda, "Code Converter Circuitry System for Selectively Rotating a Video Display Picture," U.S. Patent 4,225,929, 1980.

Received November 5, 1985; accepted for publication August 19, 1986

Karen L. Anderson *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Ms. Anderson is a staff engineer at the IBM Thomas J. Watson Research Center. Since joining IBM in 1980, she has worked in the Image Technologies Department of the Research Division, originally on the development of programming tools and later on techniques for binary image manipulation. She received an IBM Outstanding Innovation Award for her contributions to the Image View Facility in 1985. Ms. Anderson graduated from Duke University, Durham, North Carolina, with a B.S. in computer science in 1980.

Frederick C. Mintzer *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Mintzer attended Princeton University and received the Ph.D. degree in electrical engineering in 1978. Also in 1978, he joined the IBM Thomas J. Watson Research Center, engaging in research on distributed digital signal processing, signal processing architectures, and data communications. In 1980, he became the manager of the Signal Processing Applications project, and continued research in these areas. In 1983, he joined the Image Technologies Department as manager of the NCI Architectures project, engaging in research on image-processing algorithms. Dr. Mintzer received an Outstanding Innovation Award for his contributions to the Image View Facility in 1985. His current research is centered on image display and print algorithms.

Gerald Goertzel *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Goertzel joined the IBM Advanced Systems Development Division in 1964. He transferred to the Research Division in 1973. In the Advanced Systems Development Division he managed a project to develop a Clinical Decision Support System and also contributed to the design of a small computer which was extensively used in channels and large devices. At Research he led an effort to create a high-level computer design system and worked in the fields of data compression and image processing. He devised and helped evolve a program-development system of particular use in developing efficient portable code for image processing. He has also participated in the development of a halftoning process to create plates for the printing of continuous-tone black and white images on an offset press from scanned and digitized data. His present activities are concerned with the printing of color images. Prior to his joining IBM, Mr. Goertzel worked at Republic Aviation (1941), taught aeronautical engineering at New York University (1942–1943), and worked on the Manhattan Project (1944) and at Oak Ridge National Laboratory (1947–1948). He taught physics at New York University from 1948 to 1953. During this period he helped found, consulted for, and was a director of the Nuclear Development Corporation of America. In 1953 he joined that company as a full-time employee, remaining there until 1960. In 1960 he and a partner formed Sage Instruments, Inc., a company which manufactured special medical and biological instruments. Mr. Goertzel received the degrees of Mechanical Engineer and Master of Science in physics from Stevens Institute of Technology, Hoboken, New Jersey, in 1940 and a Ph.D. in physics from New York University, New York, in 1947. Mr. Goertzel is a member of Sigma Xi and a Fellow of the American Physical Society, the New York Academy of Sciences, and the American Association for the Advancement of Science.

Joan L. Mitchell *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Mitchell graduated from Stanford University with a B.S. in physics in 1969. She received her M.S. and Ph.D. degrees in physics from the University of Illinois at Champaign-Urbana in 1971 and 1974, respectively. She joined the Exploratory Printing Technologies group at the IBM Thomas J. Watson Research Center immediately after completing her Ph.D., and since 1976 has worked in the field of data compression. Dr. Mitchell received IBM Outstanding Innovation Awards for two-dimensional data compression in 1978, for teleconferencing in 1982, and for the Image View Facility and resistive ribbon thermal transfer printing technology in 1985. She is co-inventor on six patents.

Keith S. Pennington *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Pennington is senior manager of the Image Technologies and Erosion Printing Studies departments at the Thomas J. Watson Research Center. He graduated with a B.Sc. in physics from Birmingham University, England, in 1957 and a Ph.D. in physics from McMaster University, Hamilton, Ontario, Canada, in 1961. He started his research career at Bell Telephone Laboratories, Murray Hill, New Jersey, where he developed the first multicolor holograms and also did research in holographic interferometry and optical information processing. He joined IBM Research in 1967 and subsequently made several contributions to the development of improved holographic materials and techniques for three-dimensional scene analysis. Dr. Pennington was appointed manager of the exploratory terminal technologies group in 1972, and in this position he initiated the work in the development of the resistive ribbon transfer printing technology and other printing technologies. He became manager of the Image Technologies Department in 1979 and has responsibility for several projects related to high-performance videoconferencing systems, document processing, and scanning systems, as well as novel high-resolution printing processes. Dr. Pennington has written three book chapters related to holography and optical information processing and during 1971–1972 served both as a participant and as a group leader for the National Academy of Sciences Undersea Warfare Committee. While at IBM, he has received two IBM Outstanding Contribution Awards, an Outstanding Innovation Award, and an Outstanding Technical Achievement Award. Dr. Pennington is a member of the Institute of Electrical and Electronics Engineers and the Optical Society of America.

William B. Pennebaker *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Pennebaker is a Research staff member at the IBM Thomas J. Watson Research Center and currently manages a group doing research in areas related to image processing and compression. He joined IBM's Research Division in 1962 and has worked in areas related to low-temperature physics, thin films, display technology, printing technology, and image processing. Dr. Pennebaker has received an Outstanding Contribution Award for work on strontium titanate films, an Outstanding Invention Award for work on silicon nitride films, and an Outstanding Innovation Award for work on image processing and compression. He has received nine IBM Invention Achievement Awards. Dr. Pennebaker received his B.S. in engineering physics from Lehigh University, Bethlehem, Pennsylvania, in 1957, and his Ph.D. in physics from Rutgers University, New Brunswick, New Jersey, in 1962. He is a member of the American Association for the Advancement of Science, the American Institute of Physics, the Institute of Electrical and Electronics Engineers, and the Society for Information Display.