Journal of Electronic Imaging 5(4), 496–506 (October 1996).

# Efficient transmission of halftones via facsimile

**Hei Tao Fung**
**Kevin J. Parker**
University of Rochester
Department of Electrical Engineering
Rochester, New York 14627
E-mail: parker@ee.rochester.edu

**Abstract.** *The conventional method for sending halftone images via facsimile machines is inefficient. The previously proposed ToneFac algorithm improves the transmission of halftone images. ToneFac represents a halftone image by mean gray values of the disjoint blocks and an error image, which records the difference between the desired halftone and the halftone generated using the mean gray values. To improve on ToneFac, we propose additional processing techniques: searching for the error-minimizing gray value for each block; quantization and coding of block values; bit switching, which transforms the error image into a more compressible image; optimal block sizing; and spurious dot filtering, which removes perceptually insignificant dots. The new algorithm is compared to other methods, including adaptive arithmetic coding, and is shown to provide improvement in bit rate. A theoretical consideration of the compression ratio from the ToneFac algorithm is also given.*


## 1 Introduction

Text and line art can be efficiently transmitted by group 3 and group 4 facsimile devices. A compression ratio of 10 to 20 is typical.[1,2] However, the current group 3 and group 4 coding schemes are inefficient in the case where a scanned gray scale image or a halftone scanned at low resolution at the transmitter needs to be transmitted and rendered as a halftone at the receiver. One approach is to halftone the input image at the receiver first and then transmit the halftone image. For halftone images, expansion rather than compression of data usually results from using the group 3 and group 4 encoding schemes.[3,4] Halftoning is the use of black and white dot patterns to give the appearance of a gray scale,[5] so halftones intrinsically consist of many short runs of white pixels and black pixels. On the other hand, the Huffman codes specified in the standard encoding schemes have been designed for text and line art, which are characterized by many long white runs and a few short black runs.[1] Therefore, group 3 and group 4 encoding schemes are inefficient for halftones. Although halftones can be transmitted using the uncompressed mode, as allowed by the fax encoding standard, in this case, any spatial redundancy is left unutilized. Also, due to the assignment of codewords for the uncompressed mode, there is a slight data expansion. Other entropy coding schemes such as adaptive arithmetic coding[6] are applicable, but they are not standard for facsimile machines. Also, even adaptive arithmetic coding does not perform well on halftone images with blue noise characteristics, which possess little dot position regularity, but this class of halftones is important due to its pleasant appearance.[5] Another coding approach is to encode the gray scale image using a lossy coding scheme such as the Discrete Cosine Transform (DCT)-based Joint Photographic Expert Group (JPEG) scheme[7] and provide halftoning at the receiver. However, the JPEG scheme is better adapted for gray scale images, and JPEG devices are not compatible with the existing base of fax units.

The ToneFac (half*tone fax c*ompression) algorithm presented in Ref. 8 aims at compressing the original gray scale image efficiently for reproduction of the desired halftone at the receiver and making use of the standard fax encoding scheme to minimize the deviation from the standard. This paper proposes some techniques to improve on the coding efficiency of ToneFac. The resulting improved ToneFac improves the coding efficiency by a factor of 2 (Ref. 9).

## 2 ToneFac

In the ToneFac algorithm (illustrated in Figure 1), a halftone mask is assumed to be known to both the transmitter and the receiver, and the desired halftone image is generated by thresholding each pixel of the original gray scale image, using the mask value at the same location as the threshold. The original image to be rendered as a halftone at the receiver is divided into nonoverlapping rectangular blocks. For each block, the mean (or median) gray value is sent. An error image, which records the difference in the desired halftone and the halftone generated using the mean gray values, is also transmitted. Since some spatial redundancy in the original image is extracted as the mean gray values, the error dots, which indicate errors between the desired binary values and the predicted binary values from the mean gray values, are usually sparse in the error image. Such an error image, with many long white runs and a few black runs (if a black dot represents an error dot), is efficiently encoded using the group 3 or group 4 encoding scheme.

Assume the input image is $M \times N$ in size. The processing steps at the transmitter are as follows:

Paper 95-011 received May 9, 1995; revised manuscript received May 6, 1996; accepted for publication May 6, 1996.
1017-9909/96/$6.00 © 1996 SPIE and IS&T.

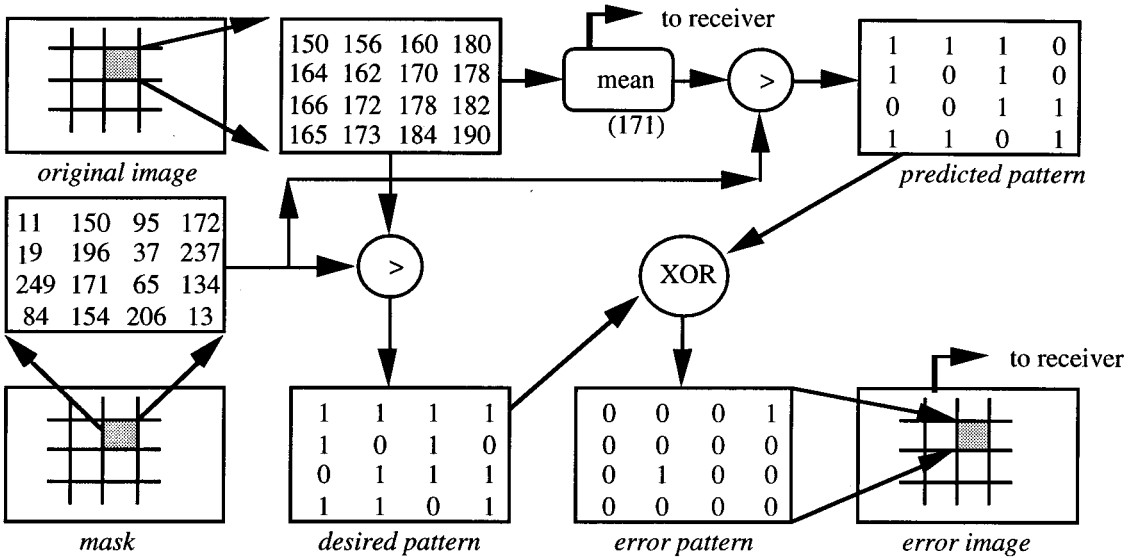


**Fig. 1** ToneFac encoding steps.

Step 1: Halftone the gray scale image $g(i,j)$ using the mask $b(i,j)$ to generate the halftone image $h(i,j)$:

$$h(i,j)=\begin{Bmatrix}0(\text{white}), & g(i,j)>b(i,j)\\ 1(\text{black}), & g(i,j)\leqslant b(i,j)\end{Bmatrix},$$
$$i=0,1,\ldots,M-1 \quad \text{and} \quad j=0,1,\ldots,N-1, \tag{1}$$

where $i$ and $j$ are indices of pixels.

Step 2: Subdivide the image into $K \times L$ blocks, where $K$ and $L$ are assumed to be factors of $M$ and $N$, respectively. If this is not the case, replications of the lines at the margins of the image may be necessary, and the redundant lines are discarded in the determination of the error image (step 4). Calculate the mean gray value for each $K \times L$ block:

$$\overline{g(m,n)}=\frac{1}{LK}\sum_{i=mK}^{mK+K-1}\sum_{j=nL}^{nL+L-1} g(i,j),$$
$$m=0,1,\ldots,\frac{M}{K}-1 \quad \text{and} \quad n=0,1,\ldots,\frac{N}{L}-1, \tag{2}$$

where $m$ and $n$ are the indices of the $K \times L$ blocks.

Step 3: Create the halftone image $\overline{h(i,j)}$ by using the block mean $\overline{g(m,n)}$ for the whole $K \times L$ block and the same halftone mask:

$$\overline{h(i,j)}=\begin{Bmatrix}0(\text{white}), & \overline{g(m,n)}>b(i,j)\\ 1(\text{black}), & \overline{g(m,n)}\leqslant b(i,j)\end{Bmatrix},$$
$$i=0,1,\ldots,M-1 \quad \text{and} \quad j=0,1,\ldots,N-1, \tag{3}$$

where $m=\text{mod}(i/K)$ and $n=\text{mod}(j/L)$.

Step 4: Create the error image $e(i,j)$ that records the inversions from black to white or from white to black needed to convert $\overline{h(i,j)}$ to $h(i,j)$:

$$e(i,j)=\overline{h(i,j)}\oplus h(i,j),$$
$$i=0,1,\ldots,M-1 \quad \text{and} \quad j=0,1,\ldots,N-1 \tag{4}$$

where $\oplus$ denotes a logical exclusive-or (XOR) operation. The 1's indicate where inversions of bits are needed. The 0's indicate no prediction errors.

Step 5: Transmit the block mean values.

Step 6: Encode and transmit error image via the group 3 or group 4 coding scheme.

Figure 1 illustrates the processing steps at the transmitter. For the purpose of illustration, we use 4×4 blocks as an example. For the original image gray values shown, the mean value is 171. Two error dots are found in the error pattern, which constitutes the error image.

At the receiver, the reconstruction process is described as follows:

Step 1: Decode for the gray mean values, and produce the corresponding halftone image:

$$\overline{h}(i,j)=\begin{Bmatrix}0(\text{white}), & \overline{g}(m,n)>b(i,j)\\ 1(\text{black}), & \overline{g}(m,n)\leqslant b(i,j)\end{Bmatrix},$$
$$i=0,1,\ldots,M-1 \quad \text{and} \quad j=0,1,\ldots,N-1, \tag{5}$$

where $m=\text{mod}(i/K)$ and $n=\text{mod}(j/L)$.

Step 2: Decode for $e(ij)$, and produce the desired halftone image $h(i,j)$:

$$h(i,j)=\overline{h}(i,j)\oplus e(i,j),$$
$$i=0,1,\ldots,M-1 \quad \text{and} \quad j=0,1,\ldots,N-1. \tag{6}$$

In the ToneFac algorithm, the compression ratio depends on the number of image blocks, the number of error dots, the halftone mask, and the coding schemes used. Unless otherwise stated, in this paper it is assumed that the blue noise mask[10,11] is used for halftoning as it generates visually pleasant halftones and is computationally efficient.

Although the compression of block values and the compression of the error image can be treated separately, the total compression ratio, which is denoted by CR, depends on both:

$$\text{CR}=\frac{Ni}{Nb+Ne}=\frac{Ni}{(Ni/\text{CR}_B)+(Ni/\text{CR}_E)}=\frac{\text{CR}_B\text{CR}_E}{\text{CR}_B+\text{CR}_E}, \tag{7}$$

where $Ni$ is the number of bits for the uncompressed halftone image, $Nb$ the number of bits for the encoded block values, $Ne$ the number of bits of the encoded error image, $\text{CR}_B$ is the ratio of the file size of the halftone image to the file size of the encoded block values, and $\text{CR}_E$ is the ratio of the file size of the halftone image to the file size of the encoded error image. With this representation, CR is less than the least of $\text{CR}_E$ and $\text{CR}_B$, and $\text{CR}_E$ and $\text{CR}_B$ are interdependent: one increases as the other decreases, and vice versa. For example, increasing the block size leads to a smaller number of block values, hence boosting $\text{CR}_B$. On the other hand, the error image records more error dots resulting from more prediction errors, hence reducing $\text{CR}_E$. Finding the optimal block size can maximize CR. Some other techniques can also be used to maximize $\text{CR}_B$ and $\text{CR}_E$ individually. We consider searching for the error-minimizing gray value for each block, and quantization and coding of block values to maximize $\text{CR}_B$. To maximize $\text{CR}_E$, we consider bit switching, which transforms the error image into a more compressible image, and spurious dot filtering, which removes perceptually insignificant dots.

## 3 Improved Processing of Block Values

### 3.1 *Optimal Block Values*

ToneFac uses the mean (or the median) block values. However, by searching the possible gray values for each image block we can determine the optimal gray value that minimizes the number of error dots on the error image. That is, the optimal block value for the $(m,n)$ subblock is $g_{\text{opt}}(m,n)$ if

$$\sum_{i=mK}^{mK+K-1}\sum_{j=nL}^{nL+L-1}|H[g(i,j),i,j]-H[g_{\text{opt}}(m,n),i,j]|$$
$$\leqslant \sum_{i=mK}^{mK+K-1}\sum_{j=nL}^{nL+L-1}|H[g(i,j),i,j]-H(x,i,j)|\forall x,$$
$$\text{where } H(x,i,j)=\begin{Bmatrix}0, & x>b(i,j)\\ 1, & x\leqslant b(i,j)\end{Bmatrix}. \tag{8}$$

Thus, to reduce the number of error pixels that need to be transmitted, we simply replace $\overline{g}(m,n)$ by $g_{\text{opt}}(m,n)$ in step 3 of the transmitting procedure and step 1 of the receiving procedure.

Comparing halftone patterns for gray values over the range of $x$ is more computationally intensive than finding the mean block value. However, we get a significant reduction in the number of error dots on the error image in return. Figure 2 illustrates the difference in using the mean block values versus using the optimal block values. The upper path of Figure 2 corresponds to using the mean block value for the block in the example. Using the mean block value 171 generates two error dots on the error pattern. The lower path uses the optimal block value for that particular block and the corresponding mask values. The optimal value falls in the range of 173 to 196. Any value in this range results in no error dots in the error pattern.

### 3.2 *Quantization of Block Values*

For a $K\times L$ image subblock, there are only $(KL+1)$ effective block gray values out of the 256 that can alter the predicted halftone pattern for that block, independent of the mask values for those pixel locations. Therefore, instead of using 256 codewords for the 256 block gray values that are possible for each subblock, we can use $(KL+1)$ codewords for the $(KL+1)$ effective block gray values. The first order entropy for the $(KL+1)$ codewords is generally less than that for the 256 codewords. Mathematically, we define the block index $I(m,n)$ for the $(m,n)$ subblock as

$$I(m,n)=k \text{ such that } b'(k)<g_{\text{opt}}(m,n)\leqslant b'(k+1), \tag{9}$$

where $b'(k)$ is the sequence of the $KL$ mask values plus two more numbers, $-1$ and 255, in the ascending order. Therefore, $I(m,n)$ has a range of $[0,KL]$. Depending on the mask values, $I(m,n)$ represents different ranges of optimal gray values varying from block to block. This way of using a smaller number of indices to represent the 256 possible block gray values can be viewed as nonuniform space-variant quantization. This reduces the number of symbols to be encoded.

### 3.3 *Coding of Block Values*

We can encode the index values by differential pulse code modulation (DPCM), taking advantage of the spatial redundancy from block to block. For simplicity, we use the first order difference between two neighboring blocks. Let $D(m,n)$ be the difference in index values for the $(m,n)$ subblock. We have

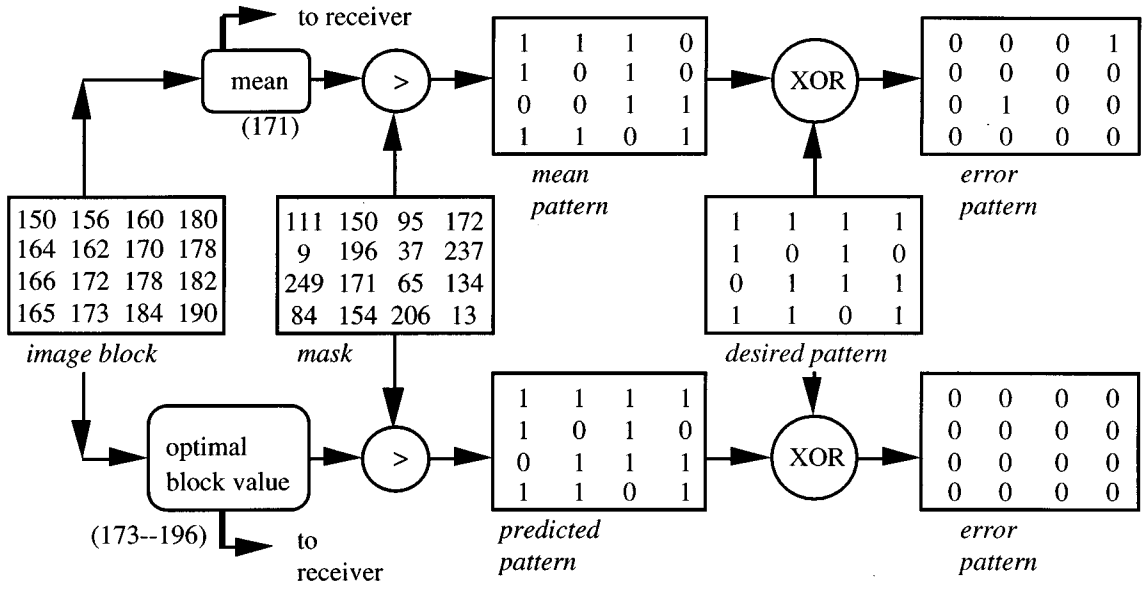


**Fig. 2** Comparison of using the mean block value and using the optimal block value for a 4×4 block.

$$D(m,n)=I(m,n)-I(m-1,n)$$

or (10)

$$D(m,n)=I(m,n)-I(m,n-1),$$

taking the difference vertically and horizontally, respectively. Each image has different orientational energy, so the direction of taking differences should be chosen to minimize the first order entropy of $D(m,n)$'s for each image. The first order entropy of $D(m,n)$'s is generally less than that of $I(m,n)$'s.

## 4 Improved Processing of Error Image

### 4.1 *Bit Switching*

The error image consists of 1's scattered on 0's. The 1's generally are the minority pixels, indicating prediction errors. Their number is image-dependent. As a rule of thumb, for group 3 and group 4, the lesser the number of runs of 1's and runs of 0's, the larger the compression ratio. Therefore, compression is enhanced if the error image is transformed line by line according to the following run-reduction rules:

1. Assume the imaginary starting pixel of each line of an error image is 0.
2. Reverse the color of the run if a 1 is encountered. Otherwise, keep the same color.

Let $y(i,j)$ be the transformed error image. Mathematically, we have

$$y(i,j)=y(i,j-1)\oplus e(i,j),$$
$$i=0,1,\ldots,M-1 \quad \text{and} \quad j=1,2,\ldots,N-1 \tag{11}$$
$$y(i,j)=0\oplus e(i,j), \quad i=0,1,\ldots,M-1 \quad \text{and} \quad j=0.$$

We call the preceding algorithm Bit Switch. The following example illustrates the transform. If a line of the error image is

0000100001010001100 0...,

this line is transformed to

0000111110011110111 1... .

It can be seen that the number of runs in the sequence is reduced from nine to six. Decoding rules are simple, and the original sequence can be recovered without error by an XOR of the adjacent pixels in the transformed image. That is,

$$e(i,j)=y(i,j-1)\oplus y(i,j),$$
$$i=0,1,\ldots,M-1 \quad \text{and} \quad j=1,2,\ldots,N-1 \tag{12}$$
$$e(i,j)=0\oplus y(i,j), \quad i=0,1,\ldots,M-1 \quad \text{and} \quad j=0.$$

Bit switching is most useful for sequences with isolated single 1's, but harmful for sequences with long runs of 1's. In most cases, bit switching is useful because the error image typically contains isolated 1's. Bit switching can be perceived as a standard way of performing runlength encoding, where we only encode the runs of 0's, and the 1's separate each run.[12] However, in our case, there would be two different sets of codewords applied to the alternate runs of 0's because group 3 and group 4 use two different sets of codewords for the white runs and the black runs.

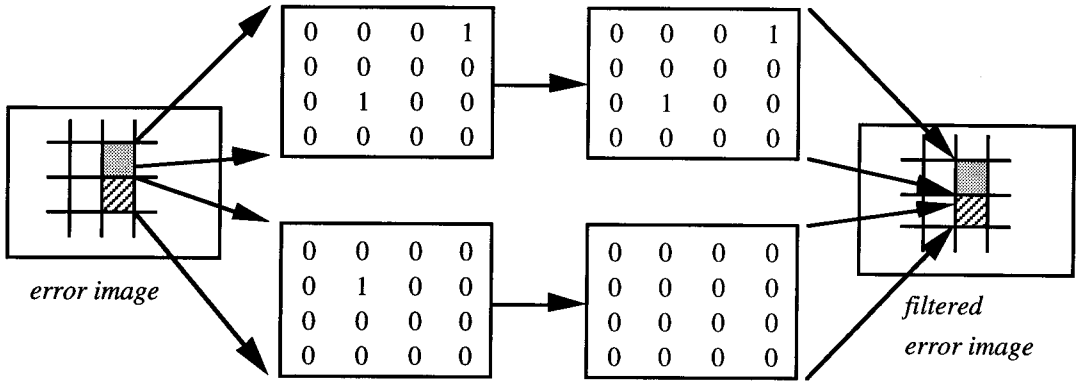


**Fig. 3** Example of spurious dot filtering on two 4×4 blocks.

### 4.2 *Optimal Block Sizing*

Increasing the block size leads to a smaller number of block values, hence boosting $CR_B$. On the other hand, the error image records more error dots, hence reducing $CR_E$. The optimal block size that maximizes CR is image-dependent and can be simply searched for over some reasonable range of block sizes. To facilitate implementation, we consider $K \times L$ rectangular blocks, where both $K$ and $L$ are powers of 2. The computational complexity depends on the block size:

$$\begin{aligned}&\text{Total number of operations}\\&=\text{number of operations per pixel per pass}\\&\quad\times\text{number of pixels per block}\\&\quad\times\text{number of passes per block}\\&\quad\times\text{number of blocks}\\&=N_p\cdot KL\cdot(KL+1)\cdot(MN/KL)\\&=N_pMN(KL+1),\end{aligned}\tag{13}$$

where $N_p$ is the number of mathematical operations such as comparisons required per pixel per pass in searching for the optimal block values. We see that the computational complexity increases linearly as the area of a block.

### 4.3 *Lossy Spurious Dot Filtering*

We call the error dots on the error image spurious dots when they contribute little visually in the final reconstructed halftone images. If we filter out these spurious dots, we can achieve a larger compression ratio.

One simple but effective filtering technique is to calculate the number of error dots in each block of the error image, where the position of the block of pixels coincides the block position on the gray scale image from which the block value is derived. For example, if 8×4 blocks on the gray scale image are used, 8×4 blocks on the error image should be used. If the number of error dots in a block does not exceed a threshold $T$, the entire block is reset to 0 (indicating no errors). The error dots that are thrown away may correspond to noise in the input image. If the number exceeds the threshold, the block is kept intact. These error dots may represent edge information, which is crucial to the quality of the halftone. That is, when the filter is applied to the $(m,n)$ subblock, we have

$$\sum_{i=mK}^{mK+K-1}\sum_{j=nL}^{nL+L-1} e(i,j)>T\Rightarrow\text{no change}$$

$$\sum_{i=mK}^{mK+K-1}\sum_{j=nL}^{nL+L-1} e(i,j)\leqslant T\Rightarrow e(i,j)=0,$$

$$i\in[mK,\ mK+K-1],\ \ j\in[nL,\ nL+L-1].\tag{14}$$

Figure 3 illustrates the case where 4×4 blocks are used, and the threshold is one. There are two error dots in the upper 4×4 error block, so no removal of error dots is done to that block. The lower error block has only one error dot, and the error dot is removed from the block. The compression ratio increases as the threshold.

## 5 Simulation Results

The effects of the proposed techniques are demonstrated in this section. Simulations are performed mainly on nine 8-bit test images as shown in Table 1. These images are

**Table 1** Test images.

| Test Image | Size |
|---|---|
| "Baboon" | 512×512 |
| "Peppers" | 512×512 |
| "Nuke" | 512×512 |
| "Lena" | 512×512 |
| "Building" | 512×512 |
| "Cablecar" | 480×512 |
| "Tanaka" | 480×512 |
| "Cameraman" | 256×256 |
| "Face" | 256×256 |

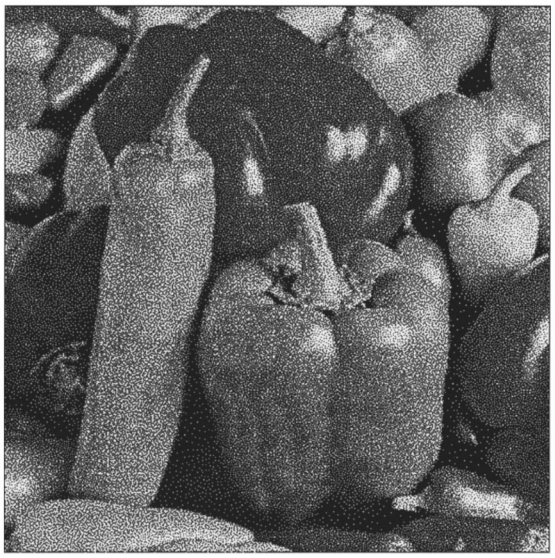

**Fig. 4** ''Peppers'' halftoned with the blue noise mask (rescreened for journal reproduction).

chosen due to their different image contents and complexity. Figure 4 shows ''Peppers'' halftoned with the blue noise mask, as an illustration in the following paragraphs.

Tables 2 and 3 underscore the gain in using the error-minimizing optimal block values over using the mean or median values as previously proposed in Ref. 8. Table 2 corresponds to the use of 8×8 blocks. On average, there is a 27.7% reduction in the number of error dots on the error image when optimal block values are used as opposed to mean block values, and compared with the use of median block values, there is a 23.7% reduction on average. Table 3 shows the results for the use of 8×4 blocks. There is an average 39.7% reduction in the number of error dots in using the optimal block values instead of the mean block values and a 36.2% reduction instead of the median block values.

The reduction in entropy of the block values by quantization and taking the first difference of block indices is shown in Table 4, where 8×8 blocks are used. The entropy of the unquantized block values is 7.217 bits/block, on average. Taking the first difference on the unquantized block values reduces the entropy to 6.353 bits/block. Quantizing the block values to 17 indices gives 5.370 bits/block. DPCM of the indices reduces the entropy to 4.557 bits/block.

**Table 2** Number of error dots in the error images when the mean, median, and optimal block values are used with a block size of 8×8.

| Test Image | Mean | Median | Optimal |
|---|---|---|---|
| ''Baboon'' | 16,753 | 16,361 | 11,944 |
| ''Peppers'' | 10,748 | 9,916 | 7,255 |
| ''Nuke'' | 36,029 | 34,029 | 28,821 |
| ''Lena'' | 9,779 | 9,383 | 6,554 |
| ''Building'' | 5,916 | 5,521 | 3,541 |
| ''Cablecar'' | 10,370 | 9,799 | 7,313 |
| ''Tanaka'' | 5,834 | 5,575 | 3,613 |
| ''Cameraman'' | 3,738 | 3,433 | 2,687 |
| ''Face'' | 3,148 | 2,917 | 2,222 |
| ''Total'' | 102,315 | 96,934 | 73,950 |

**Table 3** Number of error dots in the error images when the mean, median, and optimal block values are used with a block size of 8×4.

| Test Image | Mean | Median | Optimal |
|---|---|---|---|
| ''Baboon'' | 15,629 | 15,302 | 9,027 |
| ''Peppers'' | 8,837 | 8,051 | 4,734 |
| ''Nuke'' | 32,374 | 30,590 | 23,187 |
| ''Lena'' | 7,701 | 7,490 | 3,869 |
| ''Building'' | 5,287 | 4,875 | 2,411 |
| ''Cablecar'' | 9,625 | 9,039 | 5,584 |
| ''Tanaka'' | 4,892 | 4,664 | 2,184 |
| ''Cameraman'' | 3,217 | 2,924 | 1,953 |
| ''Face'' | 2,674 | 2,446 | 1,502 |
| Total | 90,236 | 85,381 | 54,451 |

Tables 5 and 6 show that bit switching makes the error image more compressible. Comparisons are made with the untransformed error images, one using white pixels to indicate error dots, the other using black pixels to indicate error dots. Table 5 shows the group 3 encoded file sizes in bytes, and Table 6 lists the group 4 encoded file sizes in bytes for the test images. When white pixels are used to represent error dots (1's), the compression ratios are worst. Using black pixels to represent error dots reduces the average file sizes by 23.5% for group 3 and by 16.3% for group 4, respectively. The reason is that both group 3 and group 4 cater to the statistics of long white runs and short black runs. However, there is a greater gain by using the bit switch than using a simple inversion on the error image. Group 3 on the bit switched error images reduces the average file size by 36.5% for interpreting 1's as white and 38.5% for interpreting 1's as black. Group 4 on the bit switched error images reduces the average file size by about 38.8%, where the interpretations of the 1's and 0's matter little. Figure 5 shows the error image for ''Peppers'' with error dots in black. Figure 6 shows the bit switched error image. The number of runs is reduced as can be seen.

As mentioned, the total compression ratio depends on the compression ratios for the block values and the error image. To determine the optimal block size empirically, we need to fix the coding methods for the block values and the error image. In our experiments, group 4 is used for the bit switched error image, and DPCM is used for the quantized block values. We take the difference of block indices along the direction that gives the least first order entropy. The file size for the encoded block indices is obtained through Huffman coding. The total file size is the sum of the file size for the encoded block indices and that for the group 4 encoded bit switched error image. The file size of the Huffman codebook is ignored due to its insignificant contribution: for the block size of $K \times L$, there are only $(KL + 1)$ bytes for the codebook. The encoded block indices and related information are sent through the uncompressed mode provided in the fax standard. The overhead associated with the uncompressed mode of operation is also neg-

**Table 4** Entropies (in bits per block) of the unquantized optimal block values, the first order differences of the unquantized optimal block values, the quantized optimal block values, and the first differences of the quantized optimal block values with a block size of 8×8.

| Test Image | Unquantized | First Difference, Unquantized | Quantized | First Difference, Quantized |
|---|---|---|---|---|
| ''Baboon'' | 6.958 | 6.499 | 4.991 | 4.566 |
| ''Peppers'' | 7.487 | 6.540 | 5.542 | 4.694 |
| ''Nuke'' | 7.325 | 7.329 | 5.588 | 5.439 |
| ''Lena'' | 7.382 | 6.612 | 5.432 | 4.755 |
| ''Building'' | 7.351 | 5.573 | 5.479 | 3.842 |
| ''Cablecar'' | 7.073 | 6.115 | 5.207 | 4.280 |
| ''Tanaka'' | 7.140 | 5.838 | 5.327 | 4.058 |
| ''Cameraman'' | 6.942 | 6.029 | 5.107 | 4.465 |
| ''Face'' | 7.295 | 6.641 | 5.656 | 4.918 |
| Average | 7.217 | 6.353 | 5.370 | 4.557 |

**Table 5** Fize sizes (in bytes) of the encoded error images, without and with bit switching, interpreting 1 as a white pixel and as a black pixel with the group 3 coding scheme.

| | Without Bit Switch | | With Bit Switch | |
|---|---|---|---|---|
| Test Image | (1=white) | (0=white) | (1=white) | (0=white) |
| ''Baboon'' | 23,674 | 17,852 | 13,482 | 13,154 |
| ''Peppers'' | 14,890 | 11,206 | 9,558 | 9,410 |
| ''Nuke'' | 34,566 | 28,634 | 22,150 | 21,902 |
| ''Lena'' | 14,908 | 10,834 | 9,434 | 9,048 |
| ''Building'' | 9,820 | 7,062 | 6,710 | 6,264 |
| ''Cablecar'' | 14,896 | 11,156 | 9,356 | 8,954 |
| ''Tanaka'' | 9,962 | 7,094 | 6,782 | 6,436 |
| ''Cameraman'' | 4,814 | 3,748 | 3,444 | 3,250 |
| ''Face'' | 5,010 | 3,794 | 3,312 | 3,108 |
| Total | 132,540 | 101,380 | 84,228 | 81,526 |

**Table 6** File sizes (in bytes) of the encoded error images, without and with bit switching, interpreting 1 as a white pixel and as a black pixel with the group 4 coding scheme.

| | Without Bit Switch | | With Bit Switch | |
|---|---|---|---|---|
| Test Image | (1=white) | (0=white) | (1=white) | (0=white) |
| ''Baboon'' | 24,348 | 20,110 | 13,184 | 13,144 |
| ''Peppers'' | 14,120 | 11,572 | 8,778 | 8,768 |
| ''Nuke'' | 35,068 | 31,748 | 22,838 | 22,768 |
| ''Lena'' | 14,052 | 11,166 | 8,502 | 8,454 |
| ''Building'' | 8,712 | 6,746 | 5,542 | 5,460 |
| ''Cablecar'' | 14,314 | 11,748 | 8,614 | 8,546 |
| ''Tanaka'' | 8,784 | 6,774 | 5,726 | 5,682 |
| ''Cameraman'' | 4,150 | 3,548 | 2,906 | 2,866 |
| ''Face'' | 4,526 | 3,738 | 2,756 | 2,722 |
| Total | 128,074 | 107,150 | 78,846 | 78,410 |

ligible. For ease of implementation, we consider only block sizes of $K \times L$, where $K$ and $L$ are powers of 2.

The specific results for block sizes of 4×2, 4×4, 4×8, 8×4, 8×8, and 16×16 are shown in Table 7. The sums of file sizes over the test images are shown in Figure 7, which also provides a pictorial representation of the proportion of the file size of error images to the file size of encoded block indices. As the block size increases, the file size of error images occupies a larger proportion of the total file size. From Figure 7 and Table 7, we see that 8×4 blocks give a 1.5% smaller file size than 4×8 blocks do for the test images. This is due to the fact that most natural images have more vertical edges than horizontal ones. Among all, block sizes of 4×4 and 8×4 are optimal. Considering the computational complexity, 4×4 blocks are preferred. However, further tests on the sensitivity of the optimal block sizes to resolution of images are shown in Tables 8 and 9. With decimated images, the advantage of 4×4 blocks is slightly

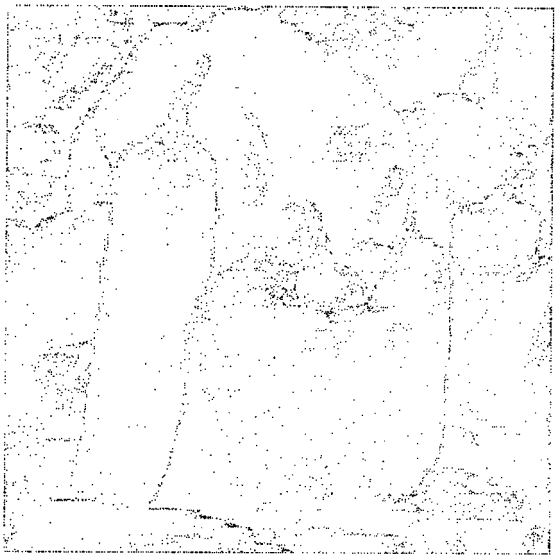

**Fig. 5** Error image for ''Peppers'', where the block size of 8×4 is used. White represents 0.

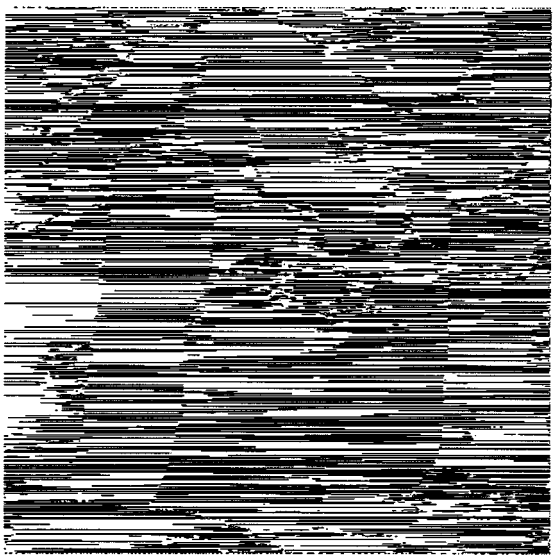

**Fig. 6** Bit switched error image of ''Peppers,'' where the block size of 8×4 is used. White represents 0.

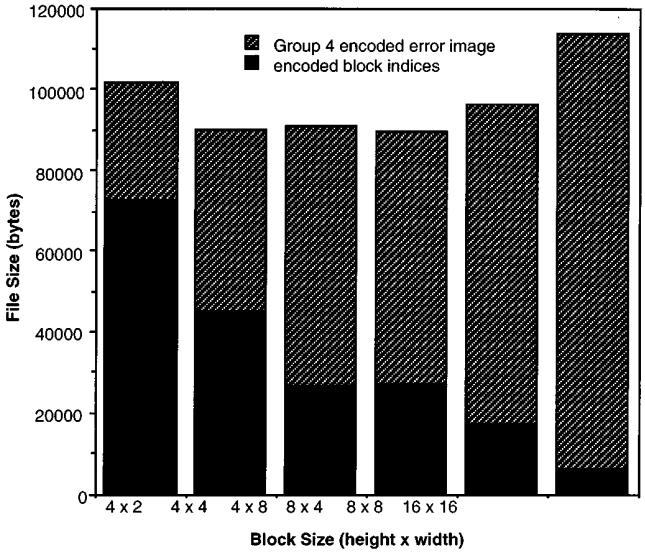


**Fig. 7** Sum of total file sizes for the nine test images versus the block size.

**Table 7** Total file size in bytes (the file size of the group 4 encoded bit switched error image and the encoded quantized optimal block values) versus the block size.

| Test Image | 4×2 | 4×4 | 4×8 | 8×4 | 8×8 | 16×16 |
|---|---|---|---|---|---|---|
| ''Baboon'' | 15,304 | 14,189 | 14,844 | 14,625 | 15,501 | 17,314 |
| ''Peppers'' | 12,141 | 10,206 | 10,502 | 10,049 | 11,187 | 14,477 |
| ''Nuke'' | 22,893 | 23,446 | 24,539 | 24,200 | 25,568 | 27,845 |
| ''Lena'' | 11,805 | 10,166 | 10,365 | 9,683 | 10,905 | 13,799 |
| ''Building'' | 10,561 | 7,744 | 7,097 | 7,307 | 7,532 | 9,591 |
| ''Cablecar'' | 12,006 | 10,323 | 10,183 | 10,427 | 10,681 | 12,192 |
| ''Tanaka'' | 9,298 | 7,171 | 7,076 | 6,971 | 7,693 | 9,954 |
| ''Cameraman'' | 3,926 | 3,499 | 3,333 | 3,300 | 3,480 | 4,027 |
| ''Face'' | 3,370 | 3,023 | 3,067 | 3,093 | 3,431 | 4,347 |
| Total | 101,304 | 89,767 | 91,006 | 89,655 | 95,978 | 113,546 |

**Table 8** Total file size (in bytes) versus the block size for some decimated test images.

| Test Image | 4×4 | 8×4 | 8×8 |
|---|---|---|---|
| ''Lena'' (256×256) | 3,581 | 3,471 | 3,951 |
| ''Peppers'' (256×256) | 3,540 | 3,586 | 3,960 |
| ''Building'' (256×256) | 2,509 | 2,589 | 2,679 |
| Total | 9,630 | 9,646 | 10,590 |

**Table 9** Total file size (in bytes) versus block size for some interpolated test images.

| Test Image | 4×4 | 8×4 | 8×8 |
|---|---|---|---|
| ''Lena'' (1024×1024) | 30,291 | 27,786 | 32,041 |
| ''Peppers'' (1024×1024) | 30,282 | 28,437 | 31,065 |
| ''Building'' (1024×1024) | 23,999 | 20,989 | 21,529 |
| Total | 84,572 | 77,212 | 84,635 |

enhanced (see Table 8). However, with interpolated images, a block size of 8×4 outperforms the others by a significant margin. Since halftoning is generally applied to low-detail images such as the interpolated test images, an 8×4 block size is preferred.

Lossy spurious dot filtering can be applied to the error image before bit switching. If the number of error dots for a block is less than or equal to a threshold, the whole block is set to 0's, indicating no prediction errors. Otherwise, the block is unchanged. Subjective tests suggest that setting the threshold to 1 for the block size of 8×4 leaves the reconstructed halftone visually indistinguishable from the original halftone. Increasing the threshold value blurs the edges more and more since the error dots corresponding to the edge information are discarded. More details are given in Table 10. Table 11 shows that there is an average 16.5% reduction in the file size of the group 4 encoded bit switched error image when the threshold is set to 1 for 8×4

**Table 10** Results of spurious dot filtering for ''Tanaka'' and ''Peppers'' with thresholds 0, 1, and 2, comparing the number of error dots on the error image and the file size (in bytes) of the group 4 encoded bit switched error image.

| | | ''Tanaka'' | |
|---|---|---|---|
| Threshold | Number of Error Dots | Group 4 Encoded Error Image (in bytes) | Comments on Recovered Halftones |
| 0 | 2184 | 4018 | Desired quality |
| 1 | 1286 | 2650 | Good; flat area affected |
| 2 | 824 | 1812 | Fair; edges also affected |
| | | ''Peppers'' | |
| Threshold | Number of Error Dots | Group 4 Encoded Error Image (in bytes) | Comments on Recovered Halftones |
| 0 | 4734 | 6444 | Desired quality |
| 1 | 3402 | 4780 | Good; flat area affected |
| 2 | 2332 | 3514 | Fair; edges also affected |

**Table 11** File sizes (in bytes) of the group 4 encoded bit switched error image with different thresholds for the spurious dot filtering.

| Test Image | Threshold=0 | Threshold=1 | Threshold=2 |
|---|---|---|---|
| ''Baboon'' | 10,830 | 8,376 | 5,170 |
| ''Peppers'' | 6,444 | 4,780 | 3,514 |
| ''Nuke'' | 19,556 | 18,250 | 16,614 |
| ''Lena'' | 5,980 | 4,258 | 2,998 |
| ''Building'' | 4,148 | 2,668 | 1,756 |
| ''Cablecar'' | 7,080 | 5,430 | 4,062 |
| ''Tanaka'' | 4,018 | 4,780 | 3,514 |
| ''Cameraman'' | 2,364 | 2,038 | 1,818 |
| ''Face'' | 2,148 | 1,678 | 1,232 |
| Total | 62,568 | 52,258 | 40,678 |

blocks. Setting the threshold to 2 gives 35.0% reduction on average. Figure 8 shows the filtered error image for ''Peppers'' as a comparison with Figure 4. Figure 9 is the reconstructed halftone as a comparison with Figure 3.

Furthermore, the improved ToneFac is compared with its old version and with adaptive arithmetic coding (Table 12). Group 4 on the original halftones results in expansion by a factor 2.29, on average. The glorified version of adaptive arithmetic coding implemented according to JBIG gives[13] a CR of 1.30:1, on average. Apparently, the blue noise patterns render the prediction ineffective. The old ToneFac refers to using unencoded mean block values and using group 4 on the error image with error dots in black.[8] Its average performance is similar to that of adaptive arithmetic coding. However, with careful analysis of Table 12, we find that the old ToneFac actually performs better (with the exception of ''Nuke''), giving a 16% smaller average file size compared with adaptive arithmetic coding. The

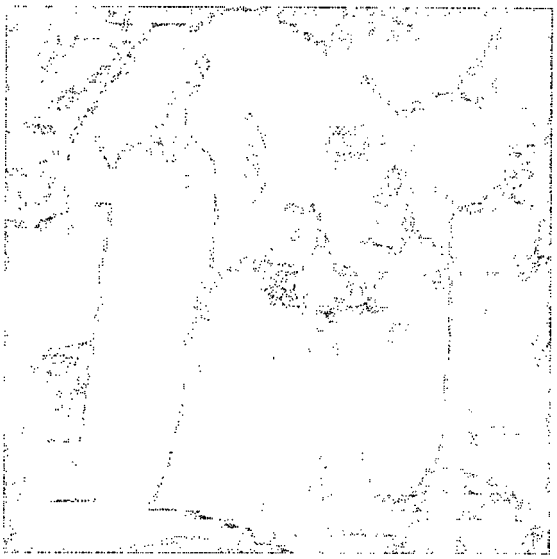

**Fig. 8** Error image for ''Peppers'' after spurious dot filtering with threshold set to 1, white represents 0, and 8×4 blocks are used.

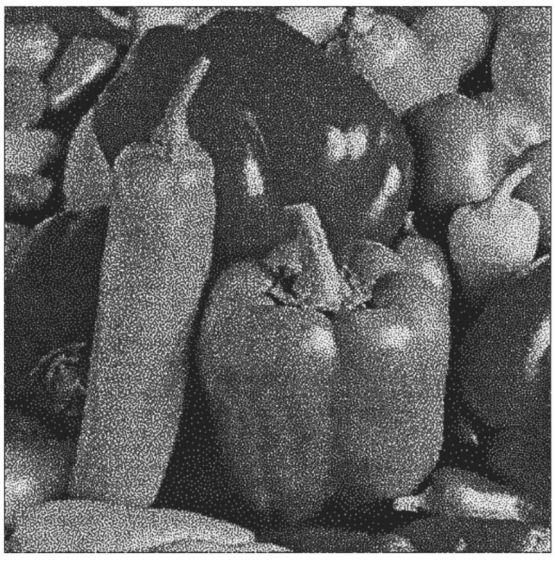

**Fig. 9** Reconstructed halftone of ''Peppers'' after spurious dot filtering with threshold set to 1 for 8×4 blocks (rescreened for journal reproduction).

**Table 12** Total file sizes (in bytes) of the halftones, which are uncompressed, encoded by the group 4 coding scheme, encoded by the adaptive arithmetic coding, encoded by the old ToneFac, and encoded by the new ToneFac. The blue noise mask is used.

| Test Image | Binary, Uncompressed | Group 4 | Arithmetic Coding | Old ToneFac | New ToneFac |
|---|---|---|---|---|---|
| ''Baboon'' | 32,768 | 84,672 | 29,015 | 32,144 | 14,625 |
| ''Peppers'' | 32,768 | 80,630 | 25,667 | 22,062 | 10,049 |
| ''Nuke'' | 32,768 | 55,612 | 23,660 | 47,554 | 24,200 |
| ''Lena'' | 32,768 | 85,896 | 27,411 | 21,124 | 9,683 |
| ''Building'' | 32,768 | 71,382 | 27,432 | 15,304 | 7,307 |
| ''Cablecar'' | 30,720 | 80,334 | 21,027 | 21,172 | 10,427 |
| ''Tanaka'' | 30,720 | 58,060 | 18,629 | 14,818 | 6,971 |
| ''Cameraman'' | 8,192 | 18,462 | 6,329 | 6,428 | 3,300 |
| ''Face'' | 8,192 | 18,010 | 6,103 | 6,366 | 3,093 |
| Average | 26,852 | 61,451 | 20,586 | 20,775 | 9,962 |

new ToneFac shows another leap in performance. It includes finding optimal block values, quantization of block values, DPCM of block indices, bit switch, and using $8\times4$ blocks. No spurious dot filtering is used. It gives an average CR of 2.70:1. Again without considering ''Nuke,'' the new ToneFac increases the average compression offered by adaptive arithmetic coding by a factor 2.47. The performance of the new ToneFac is comparable to that of adaptive arithmetic coding when Bayer's mask is used (as shown in Table 13). Without ''Nuke,'' the new ToneFac would give a 9% smaller average file size than adaptive arithmetic coding. The improvement in the performance of adaptive arithmetic coding is due to the regularity of the mask values. On the other hand, group 4 is less efficient for Bayer's halftones than for blue noise halftones because the number of runs increases in the former. For clustered dot halftones, group 4 offers a compression ratio of 1:53:1. The new ToneFac gives a 21% better CR than adaptive arithmetic coding, on average, even including ''Nuke'' (see Table 14).

Finally, we comment on the approach where the original image is compressed using a lossy encoding scheme such as JPEG and halftoning is applied to the decompressed image. The resulting halftone is generally different from the desired halftone because some information about the original image has been altered. For example, a JPEG image compressed at a low bit rate may exhibit severe block artifacts. If ordered dithering is applied, the halftone will also exhibit blockiness. Figure 10 is the ''Peppers'' image halftoned with Bayer's mask, as obtained using ToneFac with a file size of 9153 bytes. Figure 11 is the ''Peppers'' image compressed with JPEG to 9191 bytes and halftoned with Bayer's mask. The latter shows objectionable block artifacts.

## 6 Conclusions

We have improved the previously proposed ToneFac algorithm. We added a few processing techniques to the algorithm: finding optimal block values, quantizing block values to block indices, DPCM of block indices, bit switching the error image, and finding the optimal block size. With the additional processing, the algorithm gives a CR of about 3:1 for halftones generated using clustered dot masks, Bayer's masks, and blue noise masks. It is extremely efficient on blue noise halftones, outperforming adaptive arithmetic coder (baseline JBIG) by a factor of 2.

**Table 13** Total file sizes (in bytes) of the halftones, which are uncompressed, encoded by the group 4 coding scheme, encoded by the adaptive arithmetic coding, encoded by the old ToneFac, and encoded by the new ToneFac. The Bayer mask is used.

| Test Image | Group 4 | Arithmetic Coding | New ToneFAc |
|---|---|---|---|
| ''Baboon'' | 106,076 | 11,848 | 14,092 |
| ''Peppers'' | 100,668 | 10,499 | 9,153 |
| ''Nuke'' | 67,374 | 16,273 | 23,033 |
| ''Lena'' | 105,642 | 10,893 | 8,562 |
| ''Building'' | 77,876 | 9,410 | 5,980 |
| ''Cablecar'' | 91,918 | 7,735 | 9,593 |
| ''Tanaka'' | 71,958 | 8,868 | 5,976 |
| ''Cameraman'' | 23,158 | 2,692 | 3,081 |
| ''Face'' | 21,936 | 2,811 | 2,972 |
| Average | 74,068 | 9,004 | 9,161 |

**Table 14** Total file sizes (in bytes) of the halftones, which are uncompressed, encoded by the group 4 coding scheme, encoded by the adaptive arithmetic coding, encoded by the old ToneFac, and encoded by the new ToneFac. The cluster dot mask is used.

| Test Image | Group 4 | Arithmetic Coding | New ToneFac |
|---|---|---|---|
| ''Baboon'' | 22,930 | 13,629 | 13,264 |
| ''Peppers'' | 21,304 | 11,324 | 7,932 |
| ''Nuke'' | 24,332 | 16,458 | 22,005 |
| ''Lena'' | 21,556 | 11,967 | 8,084 |
| ''Building'' | 19,266 | 9,577 | 5,331 |
| ''Cablecar'' | 18,872 | 10,288 | 8,401 |
| ''Tanaka'' | 19,092 | 8,791 | 4,869 |
| ''Cameraman'' | 5,408 | 2,817 | 2,873 |
| ''Face'' | 5,036 | 6,103 | 2,551 |
| Average | 17,533 | 10,112 | 8,368 |

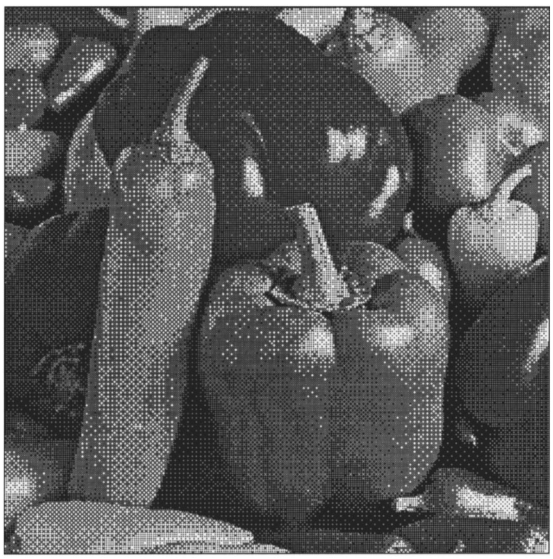

**Fig. 10** ''Peppers'' halftoned with Bayer's mask.

**Fig. 11** ''Peppers'' halftoned with Bayer's mask after JPEG compression and decompression with a quality factor of 11.

With spurious dot filtering, CRs can be increased by more than 10% depending on how harshly the error dots are filtered. The algorithm is compatible with the existing fax standards so that any addition or modification to the fax system can be minimal. Also, it can transmit the halftones derived from their original images without loss.

## Acknowledgment

This work was supported by the National Science Foundation, New York State, Center for Electronic Imaging Systems.

## References

1. R. Hunter and A. H. Robinson, ''International digital facsimile coding standards,'' *Proc. IEEE* **68**(7), 854–867 (1980).
2. S. J. Urban, ''Review of standards for electronic imaging for facsimile systems,'' *J. Electron. Imaging* **1**(1), 5–21 (1992).
3. CCITT, *Recommendation T.4, Standardization of Group 3 facsimile apparatus for document transmission*, Vol. VII-Fascicle VII.3, 21–47.
4. CCITT, *Recommendation T.6, Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus*, Vol. VII-Fascicle VII.3, 48–57.
5. R. A. Ulichney, *Digital Halftoning*, MIT Press, Cambridge (1987).
6. W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps, ''An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder,'' *IBM J. Res. Develp.* **32**(6), 717–726 (1976).
7. Digital Compression and Coding of Continuous-tone Still Images, Part I, Requirements and Guidelines. *ISO/IEC JTC1 Draft International Standard 10918-1* (Nov. 1991).
8. K. J. Parker and A. C. Cheung, ''Efficient fax transmission of halftone images,'' *J. Electron. Imaging* **1**(2), 203–208 (1992).
9. H. T. Fung and K. J. Parker, ''Improved fax transmission of halftone images,'' *Proc. SPIE* **2418**, 221–228 (1995).
10. T. Mitsa and K. J. Parker, ''Digital halftoning using a blue noise mask,'' *Proc. SPIE* **1452**, 47–56 (1991).
11. M. Yao and K. J. Parker, ''Modified approach to the construction of a blue noise mask,'' *J. Electron. Imaging* **3**(1), 92–97 (1994).
12. S. W. Golomb, ''Run length encodings,'' *IEEE Trans. Inform. Theory* **IT-12**(3), 399–401 (1966).
13. ISO/IEC JTC1/SC2/WG9, *Progressive Bi-Level Image Compression, Revision 4.1, CD 11544.*

**Hei Tao Fung** received the BS and MS degrees in electrical engineering from the University of Rochester, New York, in 1992. He completed the PhD degree in electrical engineering at the University of Rochester in 1995. His interests include image processing and integrated circuit design. Dr. Fung is a member of Tau Beta Pi and Phi Beta Kappa. Currently, he is employed at Samsung Corporation in California.

**Kevin J. Parker** received the BS degree in engineering science, summa cum laude, from the State University of New York at Buffalo in 1976. His graduate work in electrical engineering was completed at the Massachusetts Institute of Technology, with MS and PhD degrees received in 1978 and 1981. From 1981 to 1985 he was an assistant professor of electrical engineering at the University of Rochester; currently he holds the title of professor of electrical engineering and radiology. Dr. Parker has received awards from the National Institute of Medical Sciences (1979), the Lilly Teaching Endowment (1982), the IBM Supercomputing Competition (1989), and the World Federation of Ultrasound in Medicine and Biology (1991). He is a member of the IEEE Sonics and Ultrasonics Symposium Technical Committee and serves as reviewer and consultant for a number of journals and institutions. He is also a fellow of the IEEE and is on the board of governors of the American Institute of Ultrasound in Medicine. Dr. Parker's research interests are in medical imaging, linear and nonlinear acoustics, and digital halftoning.