# Emerging Standards for Still Image Compression: A Software Implementation and Simulation Study

F. Pollara and S. Arnold
Communications Systems Research Section

*This article describes the software implementation of an emerging standard for the lossy compression of continuous-tone still images. This software program can be used to compress planetary images and other two-dimensional instrument data. It provides a high-compression image-coding capability that preserves image fidelity at compression rates competitive or superior to most known techniques. This software implementation confirms the usefulness of such data compression and allows its performance to be compared with other schemes used in deep-space missions and for database storage.*

## I. Introduction

The Joint Photographic Experts Group of the International Standards Organization, together with the International Consultative Committee for Telephone and Telegraph, is in the process of developing an international standard for still image compression with both transmission and storage applications [1]. In its baseline version, the proposed algorithm consists of an $8 \times 8$ discrete cosine transform (DCT), coefficient quantization, and entropy coding (Huffman or arithmetic). The complete encoder/decoder system is illustrated by the block diagram in Fig. 1. This scheme provides a lossy high-compression image coding capability that preserves image fidelity at compression rates competitive or superior to most known techniques [2]. Its software implementation is discussed in the following section.

## II. Structure of the Software Implementation

Image samples, or pixels, are read from the original image file and sent to a two-dimensional DCT module, which produces 64 coefficients that are independently and uniformly quantized with a different step-size for each coefficient. Then a one-dimensional array is formed by reading the $8 \times 8$ matrix of quantized coefficients in a zig-zag fashion. The sequence of direct current (dc) components—the first coefficient of each block—is differentially encoded, while the alternating current (ac) components are run-length encoded. Finally, some of the most significant bits of each code are further encoded with a variable-length code; the remaining bits are transmitted essentially intact.

Flow diagrams of the software encoder and decoder structures with Huffman coding are shown in Figs. 2 and 3.

Each of the operations described in the following sections has been implemented as a separate software module to allow for future testing of modified modules.

### A. Discrete Cosine Transform Module

The forward and inverse two-dimensional DCTs used in this software implementation are defined by

$$F(u,v) = C(u)C(v)\frac{1}{4}\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)$$

$$\times \cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}$$

and

$$f(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)$$

$$\times \cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}$$

where $C(k) = 1/\sqrt{2}$ if $k = 0$, and $C(k) = 1$ if $k \neq 0$. This definition is efficiently implemented by using row-column decomposition [3]. First, the 8 × 1 DCT of each data column is computed; then the transpose of the resulting matrix is stored as an intermediate result. Finally, the 8 × 1 DCT of each data row is computed to yield the desired two-dimensional DCT. This method has the advantage of considerably reducing the total number of operations required and of limiting to $(2P-1)(P-1)$ (105 for $P = 8$) the number of cosine entries to be stored permanently for a $P \times P$ transform. A fast very large-scale integration (VLSI) of this method is described in [3]. In this article, the software implementation uses floating-point representation but reduced precision versions have also been considered. Another approach to reducing the computational complexity of this step is to consider other transforms such as the Hadamard transform [4] that can be performed using integer operations with only a slight performance degradation [2].

### B. Quantization Module

The 64 coefficients produced by the forward DCT module are quantized by a uniform or constant step-size quantizer, where the fixed step size may vary from coefficient to coefficient. This is accomplished by a predefined 8 × 8 matrix specifying the step size $Q(u,v)$ for each coefficient. The DCT of 8-bit input pixels[1] produces output coefficients with an 8-times-larger range corresponding to a total of 11 bits. The quantization matrix can, of course, be adapted to satisfy different subjective quality measures or different instrument nonlinearities.

### C. Coefficient Modeling Module

Besides using two different Huffman codes for the dc coefficient—the first term in the 8 × 8 matrix of coefficients—and the ac coefficients, these two classes are also differently pre-encoded or modeled.

**1. DC Modeling.** The quantized dc coefficient is differentially pre-encoded by computing its difference with the dc term in the previous block. These differences will then be entropy coded. Their dynamic range has now increased to 12 bits.

Two-dimensional dc prediction, which uses both the previous block's dc term and that of the block above, has also been suggested to take greater advantage of pixel correlation. This feature has not yet been implemented, but it will be included in future revisions of the software.

The prediction residual is then assigned to one of 12 categories $C_i$ $(i = 0,\ldots,11)$ defined by the base-2 logarithm of the residual's absolute value. The resulting 4-bit categories are later Huffman encoded. The remaining information about residual values and sign is encoded by a simple variable-length-integer (VLI) code, in which each codeword is $C_i$ bits long [1].

**2. AC Modeling.** As a first step, the 63 ac coefficients are reordered into a one-dimensional array by reading the 8 × 8 matrix according to a predefined zig-zag scan path. This reordering ranks the coefficients in approximate order of decreasing magnitude.

The one-dimensional array of ac coefficients is modeled by run-length coding. When a nonzero coefficient is encountered, the number of zeros preceding it and its 4-bit category (one of 11) are concatenated and stored into an 8-bit word for further Huffman coding. The remaining information about the run-length/nonzero ac pair is encoded with the same VLI code discussed above. Since only 4 bits are reserved to represent a run length, only runs up to 15 consecutive zeros can be modeled. Longer runs are artificially broken by transmitting a special code for a run

[1] The software implementation described in this article is designed for easy extension to higher input data precision, up to 12 bits per sample.

length of 16 zeros. Another special code is reserved to signal the end of a block, which also prevents the propagation of eventual channel errors to subsequent blocks.

### D. Entropy-Coding Module

Entropy coding is the process that actually performs the compaction of the data by reducing statistical redundancy. Either Huffman or arithmetic coding can be used as an entropy-coding method.

**1. Huffman Coding.** The dc category and ac run-length/category pairs are Huffman coded using two different codes. These two codes are not a disjoint partition of a larger prefix code since they contain common codewords. However, they can be decoded by their relative position in the serial stream of codewords, which is known on the receiver side since an end-of-block is always followed by a dc code, which may then be followed by ac codes or by an end-of-block code. This solution allows more efficient encoding with smaller average codeword length at the expense of a slightly more complex synchronization scheme.

The present software implementation uses two default look-up tables for the two Huffman codes. The dc table contains 12 codewords, one for each possible category; the ac table contains 162 codewords, one for each combination of 16 run lengths and 10 categories[2] plus the two special codewords for end-of-block and run-length 16. The maximum length of ac codewords is constrained to 16 bits.

The dc code tables are specified by an array of 12 bytes, which contains a properly ordered list of categories corresponding to a lexicographically ordered list of codewords belonging to a prefix code, and by an array of 16 entries representing the number of codewords of each length. The ac code tables are similarly specified by an array of 162 bytes, which contains a properly ordered list of run-length/category pairs corresponding to a lexicographically ordered list of codewords of a prefix code, and by an array of 16 entries representing the number of codewords of each length. These four arrays completely specify the two codes and can be used to send code information to the decoder or to specify custom tables adapted to the particular source of interest. This software implementation can also be used to perform a two-pass encoding in which specific codes for the image to be transmitted are created by the encoder during the first pass.

**2. Arithmetic Coding.** As a higher performance alternative to Huffman coding, arithmetic coding has also been included in the standard's specification [1]. Arithmetic coding provides a one-pass scheme that dynamically adapts to image statistics. For this reason, it has a generally superior performance [2] to the nonadaptive Huffman coding chosen for the standard that requires image statistics information before coding. Furthermore, unlike Huffman coding, arithmetic coding does not always require at least one bit per data sample.

## III. Conclusions

This article described a software implementation of a DCT-based lossy compression algorithm suitable for transmitting images from deep space and for storing images in databases. This software is now available for testing and for measuring compression and reproduction-quality performance on various instrument sources of interest. The compression procedure is executed by running the encoder program, which reads the original raster-scanned image—with 8-bit-per-pixel gray-scale resolution—to produce a compressed binary-file image and to compute the compression ratio. Different compression ratios can be obtained by changing the quantization table or the arrays specifying the Huffman codes. The decoder program, in turn, reads the compressed image and produces the reconstructed image in the same format as the original. Very satisfying reproduction quality has been obtained in preliminary tests described in [2].

In the standard considered in this article, practical hardware realization issues have been carefully evaluated to yield designs suitable for VLSI implementation. A commercial VLSI chip set based on this algorithm is already available [5]. Beyond improvements in performance or reductions in complexity that may be possible for specific deep-space instrument applications, the main challenge will be to demonstrate that this algorithm can be realized in a space-qualified version.

[2] The number of categories that can actually occur is 11 (for 8-bit data plus 3-bit expansion due to DCT), but category 0 is unused since only nonzero ac coefficients need to be encoded.

## Acknowledgments

The authors acknowledge the useful suggestions of K.-M. Cheung, S. J. Dolinar, and I. Onyszchuk during the development of this software.

## References

[1] *JPEG Draft Technical Specification (Rev. 5)*, International Standards Organization/International Consultative Committee for Telephone and Telegraph (ISO/CCITT), Washington, D.C., January 15, 1990.

[2] S. J. Dolinar, K.-M. Cheung, I. Onyszchuk, S. Arnold, and F. Pollara, "Compressed/Reconstructed Test Images for CRAF/Cassini," *TDA Progress Report 42-104* (this issue), vol. October–December 1990, Jet Propulsion Laboratory, Pasadena, California, pp. 88–97, February 15, 1991.

[3] M. Sun, T. Chen, and A. Gottlieb, "VLSI Implementation of a 16 x 16 Discrete Cosine Transform," *IEEE Trans. on Circuits and Systems*, vol. 36, no. 4, pp. 610–617, April 1989.

[4] W. K. Pratt, *Digital Image Processing*, New York: Wiley & Sons, 1978.

[5] *Technical Specification Catalog*, LSI Logic Corporation, Milpitas, California, June 1990.

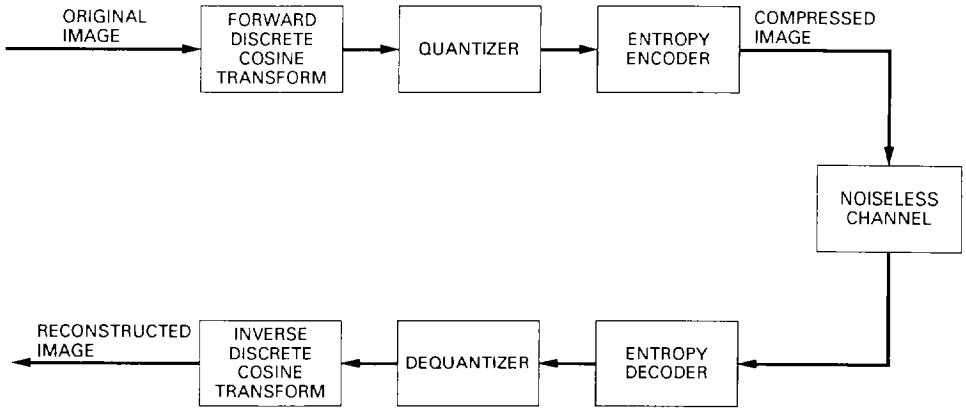


**Fig. 1. Block diagram of compression system.**

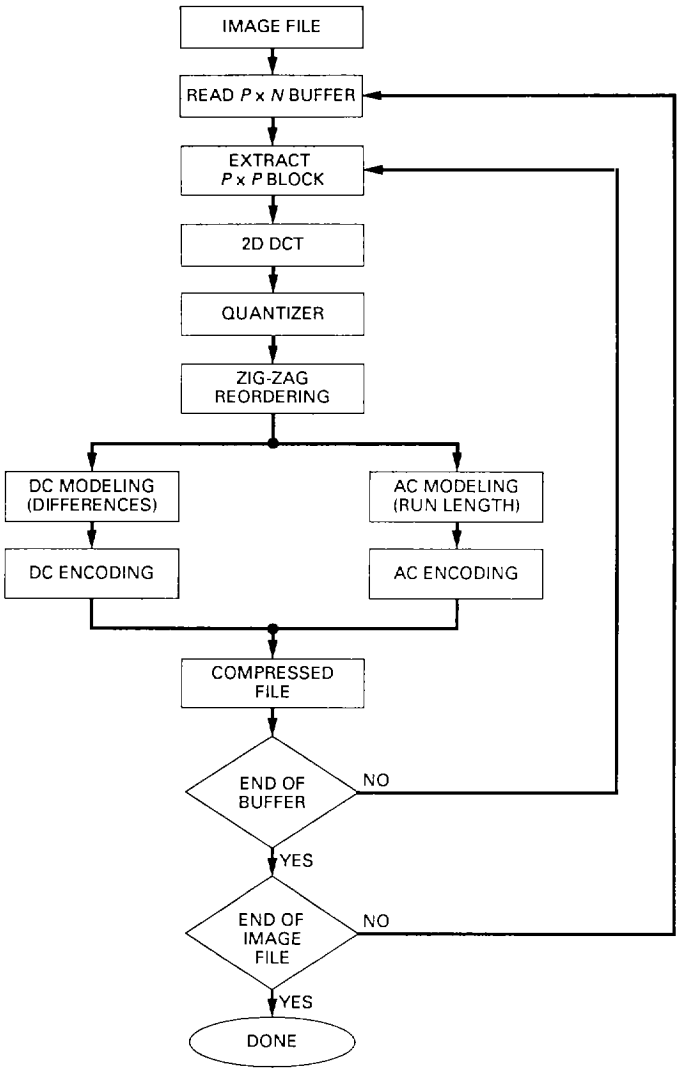


Fig. 2. Baseline compressor for Huffman coding.

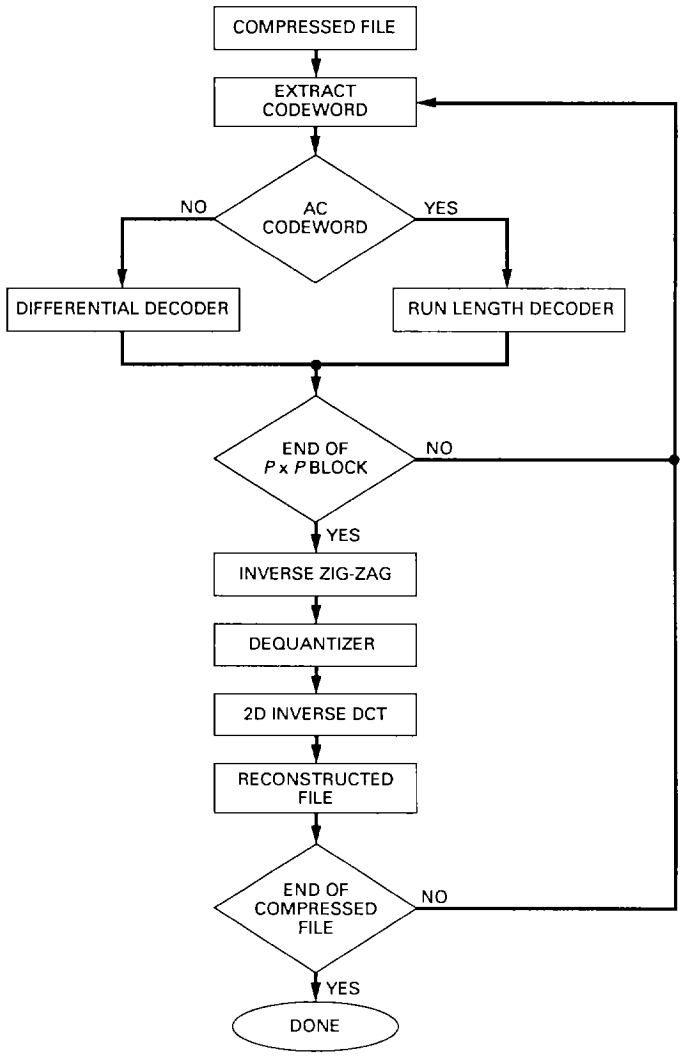


Fig. 3. Baseline decompressor for Huffman coding.