QccPack: An open-source software library for quantization, compression, and coding

James E. Fowler

Dept. of Electrical & Computer Engineering, Mississippi State University, Mississippi State, MS

ABSTRACT

We describe the QccPack software package, an open-source collection of library routines and utility programs for quantization, compression, and coding of data. QccPack is being written to expedite data-compression research and development by providing general and reliable implementations of common compression techniques. Functionality of the current release includes entropy coding, scalar quantization, vector quantization, adaptive vector quantization, wavelet transforms and subband coding, error-correcting codes, image-processing support, and general vector-math, matrix-math, file-I/O, and error-message routines. All QccPack functionality is accessible via library calls; additionally, many utility programs provide command-line access. The QccPack software package, downloadable free of charge from the QccPack Web page, is published under the terms of the GNU General Public License and the GNU Library General Public License which guarantee source-code access and as well as allow redistribution and modification. Additionally, there exist optional modules that implement certain patented algorithms. These modules are downloadable separately and are typically issued under licenses that permit only non-commercial use.

Keywords: software library, quantization, compression, coding, open source

1. INTRODUCTION

Over the last decade, the field of data compression has reached a rather advanced state of maturity, with both lossless and lossy techniques being increasingly employed in numerous real-world applications. With the advent of the World-Wide Web and its associated explosion in multimedia data, many ordinary computer users in the general public are now acutely aware of the constraints imposed by limited bandwidth and storage space. Consequently, data-compression techniques, once the exclusive domain of information theorists and communication engineers, are now increasingly employed in the day-to-day life of the typical user of today's technology. One can argue that currently we are merely at the proverbial tip of the iceberg in terms of Internet-driven multimedia use and can therefore expect continued proliferation of data compression beyond its esoteric origins. However, even the most theoretically simple compression techniques can possess complex implementations that must frequently be tailored to the application at hand; if implementations are not written for the "general case," as often research code is not, deployment to new application areas can involve painstaking modification of code "by hand" as well as lengthy and arduous debugging procedures. In order to speed adoption of data compression in new application areas, there is great need for implementations of fundamental compression techniques that are at once general, flexible, robust, and reliable. The QccPack software package, an open-source collection of library routines and utility programs for quantization, compression, and coding of data, is being written expressly for the purpose of expediting data-compression research and development by providing such general and reliable implementations of common compression techniques.

QccPack is intended for use in the development of prototypes of coding and compression systems that extend established compression theory and algorithms into new application areas. It will also be of great use in academic research that employs or builds upon fundamental compression techniques. Its main reason for existence is to obviate the incessant reimplementation of common algorithms for the sake of application-specific details. For example, a developer should not reimplement arithmetic coding every time the symbol-alphabet size changes or the number of contexts in the arithmetic-coder adaptive model is modified. Instead, QccPack allows the same implementation of arithmetic coding to be used, via a library call, regardless of application at hand. The benefits of this approach lie in the obvious efficiency associated with software reuse as well as in less obvious reliability—implementation details, which can often be complicated and not thoroughly described in academic literature, are figured out only once, for the general case, during the algorithm's implementation in QccPack. Subsequent developers can then have confidence that the implementation of a certain data-compression algorithm is correct and focus attention, instead, on higher-level application-design issues.

QccPack has been released to the public as open-source software, meaning that researchers and developers have uninhibited access to the QccPack source code. This licensing scheme is an invaluable asset for academicians seeking to deploy data-compression technology into new application areas—the developer can "peer under the hood" to see how things are done.

Additionally, although QccPack library routines are designed to be very general, it could possible that the exact functionality required by a developer is not supported by the current release of QccPack; in this case, the developer can modify, at will, the code as needed. Finally, the open-source paradigm encourages the developer to contribute such modifications and additions to the code for inclusion in future releases of QccPack so that others can benefit from his/her work as he/she has benefited from past versions of QccPack.

It should be noted that there exist a number of other software packages oriented towards compression research which are available on the Internet. For example, the Wavelet Image Compression Construction Kit by Davis *et al.*[1] implements a simple wavelet-based image coder, the full-search vector-quantization (VQ) and tree-structured VQ programs by Riskin *et al.*[2] provide two popular VQ algorithms, and the Lossless Data Compression Toolkit by de Vries *et al.*[3] includes several lossless compressors. Although toolkits such as these have a place in compression research, the goal of QccPack is typically much more ambitious, and the usefulness far greater, than that of other software packages. One of the main disadvantages of other toolkit-type software packages that they are almost always written as standalone programs, rather than libraries, and require "cut-and-paste" modification for incorporation into other projects. A cleaner interface, such as that offered by QccPack's central library of routines, is a preferable solution since the chance for incorrectly interfacing with the code is greatly reduced, and the library can be upgraded without necessitating the modification of the application code. An additional drawback to other packages is that they are usually very limited in scope and often implement only one algorithm or a small family of algorithms centered about a common theme. In contrast, QccPack provides an exceedingly large variety of functionality based upon common data structures and style of interface. Moreover, it is often the case that the source code in other software packages is several years old and is not currently being maintained or updated; however, updates to QccPack have been released more or less bimonthly over the last three years. Finally, we note that, although most software packages available in source-code form over the Internet are intended to be "free," it is usually unclear exactly what licensing scheme is in place for the software, what use is permitted by the authors, or what is the disposition of the software in regards to existing patents covering the algorithms. QccPack, on the other hand, is explicitly designated as open source, and much thought has gone into licensing issues during software development so that researchers may use QccPack with clear knowledge of the legality of their actions.

In the next section, we overview of QccPack software package, focusing on the functionality included in the current release. In subsequent sections, we discuss availability, installation, portability, and licensing issues. We then present an example illustrating the use of QccPack to accomplish a simple data-compression task—the coding of a grayscale image using full-search VQ. We conclude by discussing plans for further development of the QccPack software.

2. DESCRIPTION OF THE QCCPACK SOFTWARE

The essential component of the QccPack collection is a library (a static-linked library, `libQccPack.a`, and, if supported by the operating system, a dynamic-linked library, `libQccPack.so`) of procedures implementing a large variety of compression and coding algorithms. Application programs may make use of the QccPack library routines by linking the application against the library during compilation. Each library function is very general in its implementation so to be useful in a large variety of applications.

Additionally, much of the functionality of the library routines has been provided in the form of stand-alone executable programs. Probably the prime importance these utility programs is that they provide examples of how to interface with many of the QccPack library routines. For rapid prototyping, the utility programs can be called from shell scripts to simulate the operation of complex coding and compression systems, before implementing all the system functionality into one stand-alone program. In most cases, the utility programs provided with QccPack merely handle initialization (reading input data files, initializing needed dynamic memory, etc.) while the true heart of the processing takes place in one or more library routines.

QccPack is written entirely in ANSI C and currently consists of over 55,000 lines of source code implementing over 500 library routines and over 50 utility-program executables. The major functionalities of the library routines in the current implementation are described below.

- Entropy coding
  - Arithmetic coding[4]; includes first- and second-order adaptive models with arbitrary alphabet sizes
- Scalar Quantization (SQ)
  - Uniform SQ
  - Dead-zone SQ
  - μ-law and A-law SQ
  - Lloyd algorithm for optimal SQ design[5]

- Vector quantization (VQ)
  - Generalized Lloyd algorithm (GLA)[6] for VQ-codebook design
  - Full-search VQ encoding and decoding
  - Entropy-constrained-VQ (ECVQ)[7] training, encoding, and decoding
  - Multistage VQ (MSVQ) (also called residual VQ (RVQ))[8] training, encoding, and decoding
- Adaptive vector quantization (AVQ)
  - The generalized-threshold-replenishment (GTR) AVQ algorithm[9]
  - The Paul AVQ algorithm[10]
  - The Gersho-Yano AVQ algorithm[11]
  - Coding of side information using SQ
- Wavelet transforms, wavelet-based subband coding
  - Discrete wavelet transform (DWT) using first-generation filter banks; filter coefficients are user definable; coefficients for popular orthonormal[12] and biorthogonal[13,14] filters are included
  - Lifting implementations of DWT for popular wavelets (Daubechies 4,[12] Cohen-Daubechies-Feauveau 9/7,[13,14] Cohen-Daubechies-Feauveau 5/3 (i.e., linear lifting)[13,14]); lifting implementations are "hard-coded" into the library and run 7-10 times faster than equivalent filter-bank implementations
  - Two-dimensional DWT in the form of dyadic subband pyramids[14]
  - The stack-run (SR) algorithm[15,16] for wavelet-based subband image coding
  - The space-frequency-quantization (SFQ) algorithm[17] for wavelet-based subband image coding
- Error-correcting codes
  - Field arithmetic, including Gaussian-elimination matrix inversion
  - Reed-Solomon[18] encoding and decoding
- Image processing
  - Routines for reading and writing gray and color still images and sequences of images (via PGM and PPM formats)
  - Image and image-sequence deinterlacing
  - Differential-pulse-code modulation (DPCM) coding of images
  - Color-space conversions: RGB, YUV, CIE XYZ, CIE UCS, CIE modified UCS[19]
  - Block-based discrete cosine transform (DCT)
- General routines
  - Vector math (up/down sampling, sorting, dot product, addition/subtraction, etc.)
  - Matrix math (addition/subtraction, vector-matrix multiplication, etc.)
  - Linked lists and associated operations
  - Entropy estimation (first and second order)
  - General file input and output, including automatic detection and reading/writing of `gzip`-compressed files
  - Character bit-packing for binary bitstream input/output
  - Conversion between various file formats used by library routines
  - Error-message tracking, formatting, and output
  - Automatic command-line parameter parsing

Table 1 shows the utility programs provided by QccPack. These executable programs are provided simply for convenient command-line access to much of the commonly used QccPack functionality. In most cases, these programs merely read input files, allocate any needed space for data structures, call a QccPack library routine to perform the operation, and write output files upon completion of the library routine. For example, program `imgdpcmencode`, which provides a DPCM encoding of a grayscale image, calls library routine `QccIMGImageDPCMEncode()` to perform the actual coding. Focusing computation in library routines in this manner gives program developers easy access to all QccPack functionality.

3. DOCUMENTATION

The functionality of QccPack library routines and utility programs is documented in several ways. An eventual goal of the QccPack project is to have all library routines and utility programs documented with `man` pages that are also available in HTML form. Unfortunately, however, currently much of this documentation is incomplete—documentation exists for nearly all utility programs while about 30% of the library routines currently are documented. Completion of library-routine documentation is currently one of the top goals of the QccPack project. In addition to `man`-page documentation, and perhaps more importantly to potential QccPack developers, is the use of self-documenting code practices in the writing of the QccPack source code itself. Fig. 1 shows an excerpt of code illustrating this self-documenting philosophy—library routines and internal variables are given long, descriptive names so that persons familiar with data-compression theory can readily follow the implementation.

```
int QccIMGImageComponentDPCMEncode(QccIMGImageComponent *image_component,
                                   QccSQScalarQuantizer *quantizer,
                                   double out_of_bound_value,
                                   int predictor_code,
                                   QccIMGImageComponent *difference_image,
                                   QccChannel *channel)
{
  int row, col;
  double diff;
  int channel_symbol;
  double A, B, C, D;
  double predicted_value;
  int channel_index;
  QccIMGImageComponent reconstructed_image;
  int return_value;

  .
  .
  .

  channel_index = 0;
  for (row = 0; row < image_component->num_rows; row++)
    for (col = 0; col < image_component->num_cols; col++, channel_index++)
      {
        QccIMGImageComponentDPCMGetPreviousPixels(&reconstructed_image, row, col,
                                                  out_of_bound_value,
                                                  &A, &B, &C, &D);

        predicted_value =
          (QccIMGImageComponentDPCMPredictors[predictor_code])(A, B, C, D);

        diff = image_component->image[row][col] − predicted_value;
        if (difference_image != NULL)
          difference_image->image[row][col] = diff;

        if (QccSQScalarQuantization(diff, quantizer, NULL, &channel_symbol))
          {
            QccErrorAddMessage("(QccIMGImageComponentDPCMEncode): Error calling QccScalarQuantizat
ion()");
            goto QccError;
          }
        if (QccSQInverseScalarQuantization(channel_symbol, quantizer,
                                           &(reconstructed_image.image[row][col])))
          {
            QccErrorAddMessage("(QccIMGImageComponentDPCMEncode): Error calling QccInverseScalarQ
uantization()");
            goto QccError;
          }

        reconstructed_image.image[row][col] += predicted_value;

        if (channel != NULL)
          channel->channel_symbols[channel_index] = channel_symbol;
      }

  .
  .
  .

  return_value = 0;
  goto QccExit;
 QccError:
  return_value = 1;
 QccExit:
  QccIMGImageComponentFree(&reconstructed_image);
  return(return_value);
}
```

Figure 1. Sample code from QccPack. This routine performs DPCM on an image component (Y, U, or V component) by forming a prediction and then scalar quantizing that prediction.

Table 1. QccPack Utility Programs Listed by Functional Group

QccPack Group	*Utility Programs*		
General routines	`geometric_sequence`	`asciitodat`	`datcat`
	`arithmetic_sequence`	`datcut`	`datdist`
	`chnentropy`	`printfile`	
Entropy Coding	`chnarithmeticencode`	`chnarithmeticdecode`	
Scalar Quantization	`squniform`	`sqlloyd`	
	`sqencode`	`sqdecode`	
Vector Quantization	`gla`	`vqencode`	`vqdecode`
	`msvqtrain`	`msvqencode`	`msvqdecode`
	`ecvqtrain`	`ecvqencode`	`sqtocbk`
Adaptive VQ	`gtrencode`	`paulencode`	`gyencode`
	`gtrdecode`	`pauldecode`	`gydecode`
	`avqrate`		
Wavelets	`imgdwt`	`imgidwt`	`sbpsplit`
	`sbpassemble`	`srencode`	`srdecode`
	`sbptoicp`	`sfqencode`	`sfqdecode`
	`icptosbp`	`make_noise_image`	
Image Processing	`imgtoicp`	`icptoimg`	`imgtodat`
	`dattoimg`	`icptodat`	`dattoicp`
	`imgdist`	`seqdeinterlace`	`seqdist`
	`imgdpcmencode`	`imgdpcmdecode`	

4. LICENSING

The QccPack software package is published under the terms of the GNU General Public License (GPL) and the GNU Library General Public License (LGPL) which guarantee anyone access to the source code. In addition, the licenses permit the redistribution and modification of the source code, providing that such is done in accordance with the terms of the licenses; in essence, this requirement dictates only that all subsequent modifications and redistributions must also fall under the GPL or LGPL as appropriate. More information on the GPL and LGPL licenses can be obtained from the QccPack software itself or from the GNU Web site, `http://www.gnu.org`.

5. OPTIONAL MODULES

In addition to the functionalities listed above, there exist optional modules that can be added to the QccPack library. These modules may be available under licensing terms different from the GPL/LGPL licenses of QccPack and may contain patented algorithms. Modules exist so as to keep patented algorithms and non-GPL source code separate from the main distribution of QccPack. In this way, the main distribution remains purely open-source so that all users (commercial and non-commercial) may easily obtain and use it. The modules, on the other hand, are downloaded separately from the main distribution and are not enabled by default; the modules provide additional functionality for those users who can accept somewhat more restrictive licensing terms. At the time of this writing, the only module currently available is the QccPackSPIHT module which we describe below. Modules for other patented algorithms, such as EZW,[20] are planned and are currently under license negotiation.

5.1. The QccPackSPIHT Module

The QccPackSPIHT module is a optional module for the QccPack library that provides an implementation of the Set Partitioning in Hierarchical Trees (SPIHT)[21] algorithm for image compression. Since the SPIHT algorithm is patented, it is made available here under special license terms different from the GPL/LGPL licensing used for the main distribution of QccPack. The QccPackSPIHT license contains certain restrictions governing the terms and conditions for use, copying, distribution, and modification of the SPIHT algorithm implementation—briefly, only use in academic and non-commercial research is permitted, while all commercial use is prohibited. The QccPackSPIHT module includes two utility executables, `spihtencode` and `spihtdecode`, to perform SPIHT encoding and decoding, respectively, for grayscale images.

The SPIHT coder has been modified extensively in recent versions of the QccPackSPIHT; users of versions prior to 0.24 are strongly encouraged to upgrade. Most notably, support for arithmetic coding has been added. Additionally, the performance of the "binary-uncoded" mode (i.e., encoding without arithmetic coding) has been improved and brought closer to the performance originally reported by Said and Pearlman.[21] An extensive quantitative comparison between the QccPackSPIHT coder and the original reported results appears in the man-page documentation for the QccPackSPIHT module; to summarize, though, QccPackSPIHT is typically only 0.15 dB in PSNR below reported results in both binary-uncoded and arithmetic-coded modes.

6. AVAILABILITY, INSTALLATION, AND PORTABILITY

QccPack and optional modules are available for download, free of charge, from the QccPack Web page,

`http://qccpack.sourceforge.net`,

where one can find the current release of the source code as well as all available documentation online. QccPack is being developed on Red Hat Linux 6.1 i386; Red Hat i386 users will find both source and binary RPMs at the Web site. Users of systems other than Red Hat Linux i386 will need to download and compile the source code. However, great effort has been devoted towards making this compilation procedure as simple and as portable as possible. The steps are straightforward and consist of untarring the source archive and editing a small configuration file. Next, `imake` is invoked to generate system-specific makefiles using the supplied configuration information. Finally, a "`make install`" builds and installs the library routines, utility programs, and `man` pages. The `gcc` compiler is recommended. Although primary development efforts have concentrated on the Red Hat Linux i386 platform, it should be a straightforward procedure to build QccPack on other varieties of Linux as well as non-Linux UNIX-based systems. To date, QccPack has been compiled successfully on the following systems, although extensive debugging for these platforms has not been performed: Solaris/SPARC, Irix, HP-UX, Digital UNIX Alpha, and Digital RISC/Ultrix.

7. EXAMPLE USE

The following shows an example of using QccPack to train a VQ codebook on an image and then to code the image with full-search VQ followed with arithmetic coding. We show here how this task can be accomplished from the command line using QccPack utility programs; the same operation could be performed inside of a stand-alone C/C++ program by directly invoking appropriate QccPack library calls. We use the 512×512 grayscale Lenna image, which, along with other images and other forms of sample data, is included with the QccPack distribution.

Step 1: Convert from PGM image-file format to DAT-format file by extracting four-dimensional (2×2) vectors of pixels:
```
%> imgtodat -ts 4 lenna.pgm.gz lenna.4D.dat.gz
```
Step 2: Train a 256-codeword VQ codebook on DAT file with GLA (stopping threshold $= 0.01$):
```
%> gla -s 256 -t 0.01 lenna.4D.dat.gz lenna.4D256.cbk
```
Step 3: Vector quantize DAT file to produce channel of VQ indices:
```
%> vqencode lenna.4D.dat.gz lenna.4D256.cbk lenna.vq.4D256.chn
```
Step 4: Calculate first-order entropy of VQ indices (as bits/pixel):
```
%> chnentropy -d 4 lenna.vq.4D256.chn
First-order entropy of channel
   lenna.vq.4D256.chn
is:     1.852505 (bits/symbol)
```
Step 5: Arithmetic-encode channel of VQ indices:
```
%> chnarithmeticencode -d 4 lenna.vq.4D256.chn lenna.vq.4D256.chn.ac
Channel lenna.vq.4D256.chn arithmetic coded to:
    1.830322 (bits/symbol)
%> rm lenna.vq.4D256.chn
```
Step 6: Decode arithmetic-coded channel:
```
%> chnarithmeticdecode lenna.vq.4D256.chn.ac lenna.vq.4D256.chn
```
Step 7: Inverse VQ channel to produce quantized data:
```
%> vqdecode lenna.vq.4D256.chn lenna.4D256.cbk lenna.vq.4D256.dat.gz
```
Step 8: Convert from DAT to PGM format:
```
%> dattoimg 512 512 lenna.vq.4D256.dat.gz lenna.vq.4D256.pgm
```
Step 9: Calculate distortion between original and coded images:

```
%> imgdist lenna.pgm.gz lenna.vq.4D256.pgm
The distortion between files
   lenna.pgm.gz
         and
   lenna.vq.4D256.pgm
is:
       13.841091        (MSE)
       22.186606 dB     (SNR)
       36.719100 dB     (PSNR)
```

To compare the above results with those for SPIHT[21] coding of the same image at the same rate, we use the optional QccPackSPIHT module's encode and decode utility programs (also available as library calls). For SPIHT coding using the Cohen-Daubechies-Feauveau biorthogonal 9/7 wavelet,[13,14] periodic boundary extension, lifting wavelet implementation, five dyadic decompositions, arithmetic coding, and the same bitrate as in the VQ example above:

```
%> spihtencode -w CohenDaubechiesFeauveau.9-7.lft -b symmetric \
    -nl 5 1.830322 lenna.pgm.gz lenna.spiht.bit
SPIHT coding of lenna.pgm.gz:
  Target rate: 1.830322 bpp
  Actual rate: 1.830326 bpp
%> spihtdecode lenna.spiht.bit lenna.spiht.pgm
%> imgdist lenna.pgm.gz lenna.spiht.pgm
The distortion between files
   lenna.pgm.gz
         and
   lenna.spiht.pgm
is:
        2.530071        (MSE)
       29.566982 dB     (SNR)
       44.099476 dB     (PSNR)
```

8. CONCLUSIONS

Development of QccPack was begun in January, 1997, and continues to this day, with new versions of QccPack being released frequently (current version at the time of this writing is 0.24; check the QccPack web page for updated releases). Despite a continual state of development, most of the QccPack code has stabilized now to the point where it can be of great use to the general data-compression community, although much of the planned documentation is presently lacking. New functionality in the form of new library routines and utility programs is added with each release.

It is inevitable that QccPack users will desire certain features and functionality yet find them absent from the current release of QccPack. It is strongly hoped that, in these cases, such QccPack users will become QccPack developers by contributing additions and modifications for inclusion in future releases. The QccPack development effort welcomes all contributions. The current functionality of QccPack necessarily reflects the research interests (e.g., image/video coding) of its primary author; it is hoped that, as other researchers join the development effort, QccPack will be augmented with a broad range of techniques spanning the spectrum of current data-compression applications.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation under Grant No. INT-9600260. The work described here was initiated while the author held a postdoctoral research position at Laboratoire I3S, Université de Nice-Sophia Antipolis, Sophia Antipolis, France. Thanks go to W. Pearlman of RPI and L. Granda of PrimaComp for their aid in the licensing of SPIHT for the QccPackSPIHT module, and to SourceForge (`http://www.sourceforge.net`) for hosting the QccPack project on the Web.

REFERENCES

1. `http://www.cs.dartmouth.edu/~gdavis/wavelet/wavelet.html`.
2. `http://isdl.ee.washington.edu/COMPRESSION/code`.
3. `ftp://garbo.uwasa.fi/pc/programming/lds_11.zip`.
4. I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmetic Coding for Data Compression,” *Communications of the ACM* **30**, pp. 520–540, June 1987.
5. S. P. Lloyd, “Least-Square Quantization in PCM,” *IEEE Transactions on Information Theory* **28**, pp. 129–137, March 1982. originally an unpublished Bell Laboratories Technical note (1957).
6. Y. Linde, A. Buzo, and R. M. Gray, “An Algorithm for Vector Quantizer Design,” *IEEE Transactions on Communications* **28**, pp. 84–95, January 1980.
7. P. A. Chou, T. Lookabaugh, and R. M. Gray, “Entropy-Constrained Vector Quantization,” *IEEE Transactions on Acoustics, Speech, and Signal Processing* **37**, pp. 31–42, January 1989.
8. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer international series in engineering and computer science, Kluwer Academic Publishers, Norwell, MA, 1992.
9. J. E. Fowler, “Generalized Threshold Replenishment: An Adaptive Vector Quantization Algorithm for the Coding of Nonstationary Sources,” *IEEE Transactions on Image Processing* **7**, pp. 1410–1424, October 1998.
10. D. B. Paul, “A 500-800 bps Adaptive Vector Quantization Vocoder Using a Perceptually Motivated Distance Measure,” in *Conference Record, IEEE Globecom*, pp. 1079–1082, 1982.
11. A. Gersho and M. Yano, “Adaptive Vector Quantization by Progressive Codevector Replacement,” in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pp. 133–136, May 1985.
12. I. Daubechies, “Orthonormal Bases of Compactly Supported Wavelets,” *Communications on Pure and Applied Mathematics* **41**, pp. 909–996, November 1988.
13. A. Cohen, I. Daubechies, and J.-C. Feauveau, “Biorthogonal Bases of Compactly Supported Wavelets,” *Communications on Pure and Applied Mathematics* **45**, pp. 485–560, May 1992.
14. M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, “Image Coding Using Wavelet Transform,” *IEEE Transactions on Image Processing* **1**, pp. 205–220, April 1992.
15. M.-J. Tsai, J. D. Villasenor, and F. Chen, “Stack-Run Coding for Low Bit Rate Image Communication,” in *Proceedings of the International Conference on Image Processing*, pp. 681–684, (Lausanne, Switzerland), September 1996.
16. M.-J. Tsai, J. D. Villasenor, and F. Chen, “Stack-Run Image Coding,” *IEEE Transactions on Circuits and Systems for Video Technology* **6**, pp. 519–521, October 1996.
17. Z. Xiong, K. Ramchandran, and M. T. Orchard, “Space-Frequency Quantization for Wavelet Image Coding,” *IEEE Transactions on Image Processing* **6**, pp. 677–693, May 1997.
18. L. Rizzo, “Effective Erasure Codes for Reliable Computer Communication Protocols,” *ACM Computer Communication Review* **27**, pp. 24–36, April 1997.
19. A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1989.
20. J. M. Shapiro, “Embedded Image Coding Using Zerotrees of Wavelet Coefficients,” *IEEE Transactions on Signal Processing* **41**, pp. 3445–3462, December 1993.
21. A. Said and W. A. Pearlman, “A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees,” *IEEE Transactions on Circuits and Systems for Video Technology* **6**, pp. 243–250, June 1996.