The ICER Progressive Wavelet Image Compressor

A. Kiely[1] and M. Klimesh[1]

ICER is a progressive, wavelet-based image data compressor designed to meet the specialized needs of deep-space applications while achieving state-of-the-art compression effectiveness. ICER can provide lossless and lossy compression, and incorporates an error-containment scheme to limit the effects of data loss during transmission. The Mars Exploration Rover (MER) mission will rely primarily on a software implementation of ICER for image compression. This article describes ICER and the methods it uses to meet its goals, and explains the rationale behind the choice of methods. Performance results also are presented.

I. Introduction

In early 2004, the Mars Exploration Rover (MER) mission will land a pair of rovers on Mars. Well over half of the bits transmitted from the rovers will consist of compressed image data gathered from the unprecedented nine cameras onboard each rover. The MER rovers will rely exclusively on the ICER image compressor for all lossy image compression.

ICER is a wavelet-based image compressor designed for use with the deep-space channel. The development of ICER was driven by the desire to achieve state-of-the-art compression performance with software that meets the specialized needs of deep-space applications. ICER features progressive compression and by nature can provide both lossless and lossy compression. ICER incorporates a sophisticated error-containment scheme to limit the effects of data losses seen on the deep-space channel.

In this article, we describe ICER and the methods it uses to meet its goals, and explain the rationale behind the choice of methods. We start with an overview of ICER's features and the techniques ICER uses.

A. Progressive Compression

Under a *progressive* data compression scheme, compressed information is organized so that as more of the compressed data stream is received, reconstructed images of successively higher overall quality can be reproduced. Figure 1 illustrates this increase in quality for an example image, as a function of the resulting effective bit rate. Truncating a progressively compressed data stream by increasing amounts produces a graceful degradation in the reconstructed image quality. Thus, progressive compression provides a simple and effective method of meeting a constraint on compressed data volume without the need to guess at an appropriate setting of an image-quality parameter.
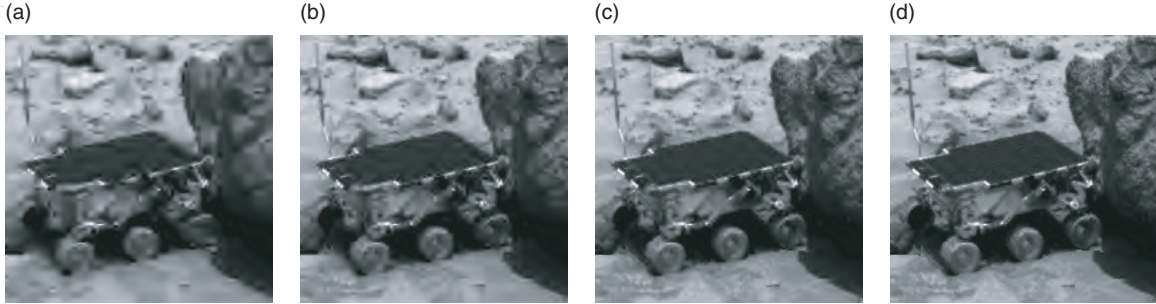

Fig. 1. This sequence of image details from a larger image shows how overall image quality improves under progressive compression as more compressed data are received: (a) 0.125 bits/pixel, (b) 0.25 bits/pixel, (c) 0.5 bits/pixel, and (d) 1 bit/pixel.

By contrast, non-progressive compression techniques typically encode information one region at a time (where a region may be a pixel or a small block of pixels), gradually covering the image spatially. Truncating the data stream produced by a non-progressive algorithm generally results in complete loss of information for some portion of the image.

Using ICER, one could send a small fraction of the data from a compressed image to get a low-quality preview, and later send more of the data to get a higher-quality version if the image is deemed interesting. In future missions, progressive compression will enable sophisticated data-return strategies involving incremental image-quality improvements to maximize the science value of returned data using an onboard buffer [1].

B. How ICER Works: An Overview

The first step in wavelet-based image compression is to apply a wavelet transform to the image. A wavelet transform is a linear (or nearly linear) transform designed to decorrelate images by local separation of spatial frequencies. The transform decomposes the image into several subbands, each a smaller version of the image, but filtered to contain a limited range of spatial frequencies.

The wavelet transforms used by ICER are described in Section II. An ICER user can select one of seven integer wavelet transforms and can control the number of subbands by choosing the number of stages of wavelet decomposition applied to the image. The wavelet transforms used by ICER are invertible; thus, image compression is lossless when all of the compressed subband data are losslessly encoded.

By using a wavelet transform, ICER avoids the "blocking" artifacts that can occur when the discrete cosine transform (DCT) is used for decorrelation, as in the Joint Photographic Experts Group (JPEG) compressor used on the Mars Pathfinder mission. The wavelet compression of ICER does introduce "ringing" artifacts (spurious faint oscillations or edges, usually near sharp edges in the image), but these tend to be less objectionable. Both types of artifacts are illustrated in Fig. 2. In addition to producing less noticeable artifacts, wavelet-based compression is usually superior to DCT-based compression in terms of quantitative measures of reconstructed image quality.

Following the wavelet transform, ICER compresses a simple binary representation of the transformed image, achieving progressive compression by successively encoding groups of bits, starting with groups containing highly significant bits and working toward groups containing less significant bits. During this encoding process, ICER maintains a statistical model that is used to estimate the probability that the next bit to be encoded is a zero. ICER's method of modeling the image is a form of *context modeling*. This and other details of the encoding procedure are described in Section III. The probability estimates produced by the context modeler are used by an *entropy coder* to compress the sequence of bits. ICER's entropy coder is described in Section IV.

(a)   (b)   (c)

Fig. 2. Details from a larger image: (a) original image, (b) reconstructed image illustrating ringing artifacts after compression to 0.125 bits/pixel using ICER, and (c) reconstructed image illustrating blocking artifacts after compression to 0.178 bits/pixel using JPEG. In this example, the ringing artifacts under ICER are less noticeable than the blocking artifacts under JPEG, even though the image is more highly compressed under ICER.

For error-containment purposes, the wavelet-transformed image is partitioned into a user-selectable number of segments, each roughly corresponding to a rectangular portion of the image. Data within each segment are compressed independently of the others so that if data pertaining to a segment are lost or corrupted, the other segments are unaffected. Increasing the number of segments (and thus reducing their size) helps to contain the effects of a packet loss to a smaller region of the image; however, it's generally harder to effectively compress smaller segments. By varying the number of segments, a user can control this trade-off between compression effectiveness and robustness to data loss, allowing some adaptability to different data loss statistics. Section V discusses error and loss mechanisms of the deep-space channel and ICER's error-containment process in more detail.

Image quality and the amount of compression are primarily controlled by two parameters: a byte quota, which controls the maximum number of compressed bytes produced, and a quality goal parameter that tells ICER to stop producing compressed bytes when a simple image-quality criterion is met. ICER stops once the quality goal or byte quota is met, whichever comes first. Section VI contains a detailed discussion of the byte quota and quality goal parameters, as well as related issues.

Section VII gives results comparing ICER's compression effectiveness to that of other compressors.

C. ICER on MER

The first space use of ICER will be on the MER mission, which will rely on a software implementation of ICER for all lossy image compression. Each MER rover will operate for 90 Martian days and will collect image data using nine visible-wavelength cameras: a mast-mounted, high-angular-resolution color imager for science investigations (the panoramic camera, or Pancam); a mast-mounted, medium-angular-resolution camera for navigation purposes (the Navcam); a set of body-mounted front and rear cameras for navigation hazard avoidance (the Hazcams); and a high-spatial-resolution camera (the Microscopic Imager) mounted on the end of a robotic arm for science investigations. With the exception of the Microscopic Imager, all of these cameras are actually stereo camera pairs. Not surprisingly, collecting and transmitting images to Earth will be a major priority of the mission.

All MER cameras produce 1024-pixel by 1024-pixel images at 12 bits per pixel. Images transmitted from MER will range from tiny 64×64 "thumbnail" images, up to full-size images. Current plans call for navigation, thumbnail, and many other image types to be compressed to approximately 1 bit/pixel, and lower bit rates (less than 0.5 bit/pixel) will be used for certain wavelengths of multi-color panoramic images. At the other end of the compression spectrum, radiometric calibration targets are likely to be compressed to about 4 to 6 bits/pixel [2]. When stereo image pairs are compressed, each image in the pair is compressed independently.

Lossless compression will be used when maximum geometric and radiometric fidelity is required, but in this case MER generally will use the LOCO (Low Complexity Lossless Compression) image compressor [3–5], which, because it is dedicated solely to lossless compression, is able to provide faster lossless compression than ICER with comparable compression effectiveness (see Section VII).

As we'll see in Section VII, ICER delivers state-of-the-art image compression effectiveness, significantly improving on the compressors used by the Mars Pathfinder mission. ICER's compression effectiveness will enhance the ability of the MER mission to meet its science objectives.[2]

II. Wavelet Transform

The first step in ICER compression is to perform a two-dimensional wavelet transform to decompose the image into a user-controlled number of subbands. Each stage of the two-dimensional wavelet decomposition is accomplished by applying a one-dimensional wavelet transform to rows and columns of data. The wavelet transform is used to decorrelate the image, concentrating most of the important information into a small number of small subbands. Thus, a good approximation to the original image can be obtained from a small amount of data. In addition, the subbands containing little information tend to compress easily. The wavelet transform does leave some correlation in the subbands, so compression of the subband data uses the predictive compression method described in Section III to attempt to exploit as much of this remaining correlation as possible.

A. One-Dimensional Wavelet Transform

Performing a one-dimensional wavelet transform on a data sequence amounts to applying a high-pass/low-pass filter pair to the sequence and downsampling the results by a factor of 2. The transform method used by ICER is from [6,7], although we use slightly different notation in our description.

We are specifically interested in reversible integer wavelet transforms. That is, we want a wavelet transform that produces integer outputs and that is exactly invertible when the input consists of integers. This allows us to achieve lossless compression when all of the transformed data are reproduced exactly.[3] Fortunately, other researchers have considered the problem of finding good reversible integer wavelet transforms; several are tabulated in [9].

An ICER user can select one of seven reversible integer wavelet transforms that are all computed in essentially the same way, but which use different choices of filter coefficients. These wavelet transforms are nonlinear approximations to linear high-pass/low-pass filter pairs. The nonlinearity arises from the use of roundoff operations.

We begin with a length-N ($N \geq 3$) data sequence $x[n]$, $n = 0, 1, \cdots, N-1$. A wavelet transform of this sequence produces $\lceil N/2 \rceil$ low-pass outputs $\ell[n]$, $n = 0, 1, \cdots, \lceil N/2 \rceil - 1$, and $\lfloor N/2 \rfloor$ high-pass outputs $h[n]$, $n = 0, 1, \cdots, \lfloor N/2 \rfloor - 1$. Thus, the total number of outputs from the wavelet transform is equal to the number of data samples.

---

[2] For more information on the usage, implementation, and operational details of the the ICER software used by MER, we refer the reader to A. Kiely and M. Klimesh, *Mars Exploration Rover (MER) Project ICER User's Guide,* JPL D-22103, MER 420-8-538 (internal document), Jet Propulsion Laboratory, Pasadena, California, December 13, 2001, which explains details such as the organization of compressed data in the encoded bitstream, limits on allowed values of ICER parameters, and input/output interfaces.

[3] It is easy to construct integer wavelet transforms that are reversible but not well suited to data compression. For example, one could use a linear filter pair with integer coefficients that have large magnitude, thus producing output with a large dynamic range. In this case, there would be a lot of bits to encode, and the least-significant bits would contain a substantial amount of redundancy; hence, the transform would not be convenient for lossless or near-lossless data compression. We are really interested in *efficient* reversible transforms; see [8, p. 214] for a discussion.

The low-pass outputs $\ell[n]$, $n = 0, 1, \cdots, \lceil N/2 \rceil - 1$ are given by

$$\ell[n] = \begin{cases} \left\lfloor \dfrac{1}{2}\left(x[2n] + x[2n+1]\right) \right\rfloor, & n = 0, 1, \cdots, \left\lfloor \dfrac{N}{2} \right\rfloor - 1 \\[3ex] x[N-1], & n = \dfrac{N-1}{2}, \; N \text{ odd} \end{cases}$$

The low-pass filter used for each of the wavelet transforms is the same, and essentially takes averages of adjacent data samples.

To compute the high-pass outputs, for $n = 0, 1, \cdots, \lceil N/2 \rceil - 1$ we first compute

$$d[n] = \begin{cases} x[2n] - x[2n+1], & n = 0, 1, \cdots, \left\lfloor \dfrac{N}{2} \right\rfloor - 1 \\[3ex] 0, & n = \dfrac{N-1}{2}, \; N \text{ odd} \end{cases} \tag{1}$$

and

$$r[n] = \ell[n-1] - \ell[n], \quad n = 1, 2, \cdots, \left\lceil \dfrac{N}{2} \right\rceil - 1 \tag{2}$$

Then for $n = 0, 1, \cdots, \lfloor N/2 \rfloor - 1$, the high-pass outputs $h[n]$ are computed as

$$h[n] = d[n] - \begin{cases} \left\lfloor \dfrac{1}{4} r[1] \right\rfloor, & n = 0 \\[3ex] \left\lfloor \dfrac{1}{4} r[1] + \dfrac{3}{8} r[2] - \dfrac{1}{4} d[2] + \dfrac{1}{2} \right\rfloor, & n = 1, \alpha_{-1} \neq 0 \\[3ex] \left\lfloor \dfrac{1}{4} r\left[\dfrac{N}{2} - 1\right] \right\rfloor, & N \text{ even}, n = N/2 - 1 \\[3ex] \left\lfloor \alpha_{-1} r[n-1] + \alpha_0 r[n] + \alpha_1 r[n+1] - \beta d[n+1] + \dfrac{1}{2} \right\rfloor, & \text{otherwise} \end{cases} \tag{3}$$

Table 1 gives the filter parameters $\alpha_{-1}, \alpha_0, \alpha_1, \beta$ for each of the filters used by ICER. Filter Q was devised by the first author of this article; the others are from [6,7]. Filter A is also essentially the same as the "Reversible Two-Six" transform in [8]; the only differences are in the way roundoff operations are performed.

The high-pass filter outputs $h[n]$ are approximately equal to the following linear filter outputs:

$$\hat{h}[n] = \sum_{i=-4}^{3} c_i x[2n+i] \tag{4}$$

where the filter coefficients are

$$(c_{-4}, c_{-3}, c_{-2}, c_{-1}, c_0, c_1, c_2, c_3) =$$

$$\frac{1}{2}(-\alpha_{-1}, -\alpha_{-1}, \alpha_{-1} - \alpha_0, \alpha_{-1} - \alpha_0, 2 + \alpha_0 - \alpha_1, -(2 - \alpha_0 + \alpha_1), \alpha_1 + 2\beta, \alpha_1 - 2\beta) \qquad (5)$$

Table 2 lists the values of these coefficients for each filter. For each filter, it can be shown that the difference between the exact and approximate high-pass filter output is bounded as follows:

$$|h[n] - \hat{h}[n]| \le \frac{1}{2} + \frac{1}{4}\left(|\alpha_{-1}| + |\alpha_0 - \alpha_{-1}| + |\alpha_1 - \alpha_0| + |\alpha_1|\right) \le \frac{25}{32} \qquad (6)$$

To invert the transform given the high-pass outputs $h[n]$ and low-pass outputs $\ell[n]$, we first compute the values of $r[n]$ using Eq. (2). Then we compute the values of $d[n]$ by inverting Eq. (3). This computation is done in order of *decreasing* index n.[4] Finally, we recover the original data sequence $x[n]$ using

$$x[2n] = \ell[n] + \left\lfloor \frac{d[n] + 1}{2} \right\rfloor$$

$$x[2n + 1] = x[2n] - d[n]$$

Reversible integer wavelet transforms such as the ones described here also can be computed using the "lifting" technique; see [9] for a good summary. Both methods require the same number of arithmetic operations. Note that roundoff operations in the transforms of [9] are done in a slightly different way; consequently, the inverse transforms are also different.

Table 1. Wavelet filter parameters.

Filter	α_{-1}	α_0	α_1	β
A	0	1/4	1/4	0
B	0	2/8	3/8	2/8
C	−1/16	4/16	8/16	6/16
D	0	4/16	5/16	2/16
E	0	3/16	8/16	6/16
F	0	3/16	9/16	8/16
Q	0	1/4	1/4	1/4

---

[4] To begin calculating the sequence of $d[n]$ values, when N is even, note from Eq. (3) that $d[(N/2) - 1] = h[(N/2) - 1] + \lfloor(1/4)r[(N/2) - 1]\rfloor$, and when N is odd, note from Eq. (1) that $d[(N - 1)/2] = 0$.

Table 2. Filter coefficients for the linear approximations to filters used by ICER.

Filter	c_{-4}	c_{-3}	c_{-2}	c_{-1}	c_0	c_1	c_2	c_3
A	0	0	$-1/8$	$-1/8$	1	-1	$1/8$	$1/8$
B	0	0	$-1/8$	$-1/8$	$15/16$	$-17/16$	$7/16$	$-1/16$
C	$1/32$	$1/32$	$-5/32$	$-5/32$	$7/8$	$-9/8$	$5/8$	$-1/8$
D	0	0	$-1/8$	$-1/8$	$31/32$	$-33/32$	$9/32$	$1/32$
E	0	0	$-3/32$	$-3/32$	$27/32$	$-37/32$	$5/8$	$-1/8$
F	0	0	$-3/32$	$-3/32$	$13/16$	$-19/16$	$25/32$	$-7/32$
Q	0	0	$-1/8$	$-1/8$	1	-1	$3/8$	$-1/8$

B. Two-Dimensional Wavelet Transform

To perform a single stage of a two-dimensional wavelet decomposition, we first perform the one-dimensional transform on each row of the image [Fig. 3(a)], replacing each row with the low-pass and high-pass output of the wavelet transform, as depicted in Fig. 3(b). Next we perform the one-dimensional transform on each column of the transformed data to arrive at the result shown in Fig. 3(c). This operation produces four spatially filtered lower-resolution versions of the image, referred to as *subbands*. We refer to the elements of both the transformed image and the original image as "pixels"; the meaning will be clear from the context.

Since ICER uses nonlinear wavelet transforms, a two-dimensional decomposition stage must be inverted in reverse order; i.e., the decompressor must first perform the inverse on the columns and then on the rows.

After a single stage of decomposition, the subband containing the lowest spatial frequencies (i.e., the output of the low-pass horizontal and low-pass vertical filtering) is essentially a lower-resolution version of the original image, obtained by averaging 2×2 blocks of pixels. Successive stages of wavelet decomposition are performed by applying the two-dimensional wavelet decomposition to the lowest-frequency subband from the previous stage. This produces the pyramidal decomposition first suggested by Mallat [10]. When an image of dimensions $W \times H$ undergoes D stages of decomposition, the lowest-frequency subband will have dimensions $\lceil W/2^D \rceil \times \lceil H/2^D \rceil$. In ICER, the user specifies the number of stages of wavelet decomposition.

The motivation for further decomposition of the lowest-frequency subband is to better exploit the high correlation in this subband. After a few (typically 3 to 6) decomposition stages, the resulting lowest-frequency subband becomes small enough that only a small portion of the compressed bitstream is devoted to it, and further decomposition stages have a negligible effect on compression effectiveness.

Each successive wavelet decomposition stage replaces the lowest-frequency subband with four new subbands; thus, after D stages of decomposition the total number of subbands is $3D + 1$. Subbands produced by the first-stage decomposition are called level-1 subbands, those produced by the second stage decomposition are called level-2 subbands, etc. We also identify subbands by LL, LH, HL, or HH, where L and H indicate low-pass and high-pass filtering, respectively; the first letter refers to the horizontal direction, and the second letter refers to the vertical direction. Figure 4 shows the ten subbands produced by three stages of wavelet decomposition along with the subband labels.
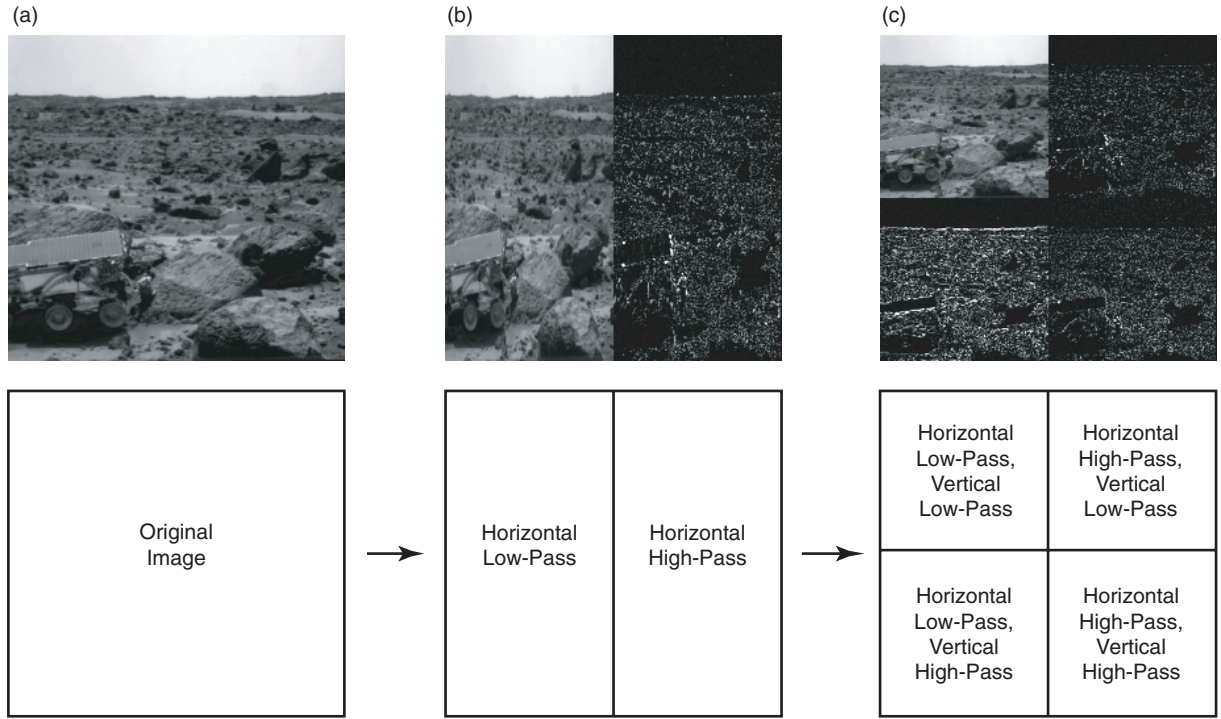
7

Fig. 3. One stage of a two-dimensional wavelet decomposition: (a) original image, (b) horizontal decomposition of the image, and (c) two-dimensional decomposition of the image. Pixel magnitudes are shown, and the image is contrast enhanced (magnitudes scaled by a factor of 6) in the subbands that have been high-pass filtered.
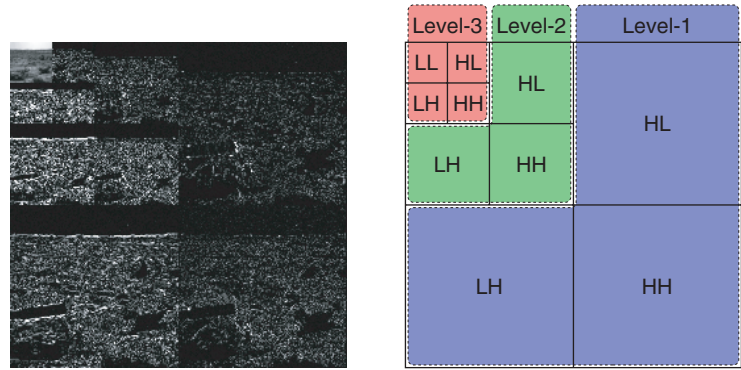

Fig. 4. The ten subbands produced by three stages of wavelet decomposition. The image is contrast enhanced (magnitudes scaled by a factor of 6) in all but the LL subband.

C. Dynamic Range of Wavelet-Transformed Data

Since the low-pass filter operation amounts to pixel averaging, the range of values that the low-pass output $\ell[n]$ can take on is the same as that of the input. The high-pass filter output, however, has an expanded dynamic range compared to the input. An overflow will occur if an output from the wavelet transform is too large to be stored in the binary word provided for it. In this case, the value stored will be incorrect, which will cause a localized but noticeable artifact in the reconstructed image.[5] To guarantee

---

[5] The compression effectiveness may suffer slightly because the incorrect transform value usually will be more difficult to predict, but otherwise compression will proceed normally.

that overflow cannot occur, the binary words used to store the wavelet transform output must be large enough to accommodate the dynamic range expansion.

For a given high-pass filter, the linear approximation, Eq. (4), suggests that the filter's maximum output value occurs when maximum input values coincide with positive filter coefficients and minimum input values coincide with negative filter coefficients. Let $h_{\max}$ denote the maximum possible output value from the high-pass filter, and let $\hat{h}_{\max}$ denote the approximation to this value derived from the linear approximation to the filter. Then, if each input value $x[n]$ can take values in the range $[x_{\min}, x_{\max}]$, we have

$$h_{\max} \approx \hat{h}_{\max} = \sum_{i:c_i>0} c_i x_{\max} + \sum_{i:c_i<0} c_i x_{\min} = x_{\min} \sum_i c_i + (x_{\max} - x_{\min}) \sum_{i:c_i>0} c_i$$

We notice from Eq. (5) that $\sum_i c_i = 0$, so

$$\hat{h}_{\max} = (x_{\max} - x_{\min}) \sum_{i:c_i>0} c_i = (x_{\max} - x_{\min})\frac{1}{2} \sum_i |c_i|$$

Similar analysis shows that for the corresponding minimum possible output value $h_{\min}$ and its approximation $\hat{h}_{\min}$, we have

$$h_{\min} \approx \hat{h}_{\min} = -\hat{h}_{\max}$$

We can use Eq. (6) to bound the accuracy of these approximations:

$$|h_{\max} - \hat{h}_{\max}| \leq \frac{25}{32}$$

$$|h_{\min} - \hat{h}_{\min}| \leq \frac{25}{32}$$

Thus, following one high-pass filter operation,

$$\frac{h_{\max} - h_{\min}}{x_{\max} - x_{\min}} \approx \sum_i |c_i|$$

and so the approximate dynamic range expansion in bits is

$$\log_2 \left(\sum_i |c_i| \right)$$

Since low-pass filtering does not change the range of possible output values, the dynamic range of a subband is determined by the number of high-pass filtering operations used to produce it. Under the pyramidal wavelet decomposition used by ICER, the HH subbands require two high-pass filtering operations (one horizontal and one vertical), while the other subbands require at most one such operation. The output dynamic range is fully utilized when the rows and columns of the image conspire together to maximize the range of output values in an HH subband.

Let $h_{\min}^{(2)}$ and $h_{\max}^{(2)}$ denote the minimum and maximum possible output values after two high-pass filter operations, and let $\hat{h}_{\min}^{(2)}$ and $\hat{h}_{\max}^{(2)}$ denote the corresponding approximations. Then

$$h_{\max}^{(2)} \approx \hat{h}_{\max}^{(2)} = -\hat{h}_{\min}^{(2)} = (\hat{h}_{\max} - \hat{h}_{\min})\frac{1}{2}\sum_i |c_i| = (x_{\max} - x_{\min})\frac{1}{2}\left(\sum_i |c_i|\right)^2 \tag{7}$$

Using Eq. (6), it can be shown that the error in this approximation satisfies

$$|h_{\max}^{(2)} - \hat{h}_{\max}^{(2)}| \le \frac{25}{32}\left(1 + \sum_i |c_i|\right) < 3.3$$

for each filter. From the approximation of Eq. (7), we have

$$\frac{h_{\max}^{(2)} - h_{\min}^{(2)}}{x_{\max} - x_{\min}} \approx \left(\sum_i |c_i|\right)^2$$

and following two high-pass filtering operations, the approximate dynamic range expansion in bits is

$$2\log_2\left(\sum_i |c_i|\right)$$

Table 3 gives the approximate dynamic range expansion following one or two high-pass filter operations, for each filter used by ICER. It can be shown that the dynamic range expansion at the edges of the subbands is less than the values given in the table.

If we use b-bit words to store the wavelet transformed image, then to guarantee that overflow does not occur we need $h_{\max}^{(2)} \le 2^{b-1} - 1$ and $h_{\min}^{(2)} \ge -(2^{b-1} - 1)$.[6] Thus, the constraint on the input is approximately

$$x_{\max} - x_{\min} \le \frac{2(2^{b-1} - 1)}{\left(\sum_i |c_i|\right)^2} \tag{8}$$

Since low-pass filtering does not expand the dynamic range, this operation only adds the requirement that the input values can be stored as b-bit words.

For a given constraint on the dynamic range of the transform output, we can use the actual nonlinear wavelet transform to determine an exact constraint on the input range. We have used this method to compute constraints on input pixel values for 8-bit, 16-bit, and 32-bit output words, with one or two high-pass filter operations; the results are given in Table 4. If the difference $x_{\max} - x_{\min}$ is less than or equal to the value indicated in the table, overflow cannot occur.

---

[6] The requirement on $h_{\min}^{(2)}$ is $h_{\min}^{(2)} \ge -(2^{b-1} - 1)$ rather than $h_{\min}^{(2)} \ge -2^{b-1}$ because the transformed pixels are converted to sign-magnitude form; see Section III.A.

Table 3. Approximate dynamic range expansion following one or two high-pass filter operations.

Filter	One high-pass filter operation		Two high-pass filter operations	
	$\sum_i |c_i|$	$\log_2\left(\sum_i |c_i|\right)$, bits	$\left(\sum_i |c_i|\right)^2$	$2\log_2\left(\sum_i |c_i|\right)$, bits
A	5/2	1.32	25/4	2.64
B	11/4	1.46	121/16	2.92
C	25/8	1.64	625/64	3.29
D	41/16	1.36	1681/256	2.72
E	47/16	1.55	2209/256	3.11
F	51/16	1.67	2601/256	3.34
Q	11/4	1.46	121/16	2.92

Table 4. Maximum input dynamic range (i.e., maximum difference $x_{max} - x_{min}$) that guarantees a given size output word will not overflow following one or two high-pass filter operations. Entries in this table are computed exactly using the nonlinear wavelet transforms.

Filter	One high-pass filter operation			Two high-pass filter operations		
	8-bit word	16-bit word	32-bit word	8-bit word	16-bit word	32-bit word
A	101	26213	1717986917	40	10485	687194766
B	92	23830	1561806289	33	8665	567929559
C	81	20971	1374389534	25	6710	439804651
D	99	25574	1676084798	38	9980	654081872
E	86	22309	1462116525	29	7594	497741795
F	79	20559	1347440719	24	6449	422726500
Q	92	23830	1561806289	33	8665	567929559

For example, if we use 16-bit words (i.e., $b = 16$) to store the output of two high-pass filter operations using filter F, then, from Table 3, the constraint on input dynamic range from Eq. (8) becomes

$$x_{max} - x_{min} \leq \frac{2(2^{16-1} - 1)}{2601/256} \approx 6450.1$$

In fact, Table 4 shows that this calculation turns out to be just slightly optimistic.

On MER, all cameras produce 12-bit pixels and each is stored using a 16-bit word. From Table 3, the worst-case dynamic range expansion following two high-pass filter operations is about 3.34 bits, arising from filter F. Thus, 16-bit words are adequate to store the output of the wavelet transform. We can confirm this result using Table 4, which shows that for each filter, using 16-bit words, we can accommodate an input pixel dynamic range ($x_{max} - x_{min}$) of at least 6449, which easily supports 12-bit input pixels. In fact, on MER we conserve onboard memory by reusing the memory array containing the original image to store the wavelet transformed image.

As another example, we observe from Table 4 that, for 14-bit input pixels, 16-bit words are adequate to store the wavelet transform output following one but not two high-pass filter operations. Thus, to

conserve memory, in principle one could use 32-bit words to store the subbands resulting from two high-pass filter operations (this represents less than a third of the transformed pixels) and 16-bit words for the other subbands.

While the output dynamic range from the high-pass filter is expanded compared to the input dynamic range, it is observed that the actual range of output values following high-pass filtering tends to be smaller than the range of original pixel values when applied to natural images. Thus, even when overflow is possible, it may be rare. As a test, we produced a "pure noise" image with pixel values independently drawn from a uniform random distribution over $[0, 2^{14} - 1]$. Following high-pass filtering in the horizontal and vertical directions using filter B, we observed overflow of 16-bit words on only about 0.07 percent of the pixels in the resulting HH subband.

D. Quantitative Filter Comparison

Figure 5 illustrates the difference in ICER's rate-distortion performance on a Mars surface image when different wavelet filters are used. Tests on other Mars surface images produced similar comparisons. However, this comparison should be interpreted with caution for two reasons: First, qualitative differences in the reconstructed images can be more significant than the small differences in a distortion metric; in fact, preliminary evidence suggests that scientists would rank the wavelet filters substantially differently than suggested by the figure. Second, the type of image may significantly affect the filter comparison; for example, remarks in [7] suggest that filter C may have an advantage on very smooth images.

Table 5 shows the effect of the choice of wavelet filter on ICER's lossless compression performance. The images tested are all 12-bit/pixel Mars surface images, so it is quite possible that the comparison will be different for other types of images. Note that the ranking suggested by this test is somewhat different from the ranking suggested by our lossy compression test.

III. Bit-Plane Coding

After the wavelet decomposition, ICER encodes a simple binary representation of the transformed image. The compressed image consists of a compressed description of some subset of the bits in this representation. The decompressor then can produce an approximation of the transformed image from
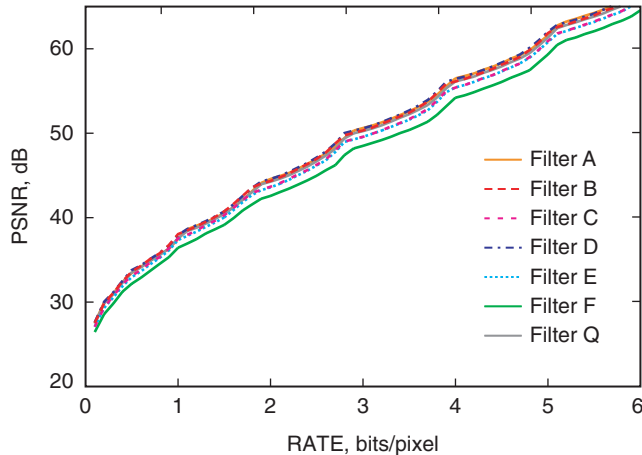

Fig. 5. Rate-distortion performance for each of the wavelet filters used by ICER. Results are for Mars surface test image _a_ (described in Section VII), compressed with one error-containment segment and four stages of decomposition. See Section VII.A for a definition of peak signal-to-noise ratio (PSNR).

Table 5. Lossless compression performance (rate in bits/pixel) for each of the wavelet filters used by ICER. Results are given for the 6 Mars surface test images described in Section VII. Images *a* through *e* were compressed using 4 stages of decomposition, and the larger image *m* was compressed using 6 stages of decomposition. A single error-containment segment was used in each case.

Image	Filter						
	A	B	C	D	E	F	Q
a	8.25	8.19	8.20	8.20	8.22	8.26	8.24
b	8.97	8.90	8.90	8.92	8.92	8.96	8.96
c	9.16	9.08	9.08	9.11	9.11	9.15	9.15
d	9.48	9.40	9.40	9.42	9.42	9.46	9.46
e	8.83	8.76	8.76	8.78	8.78	8.82	8.82
m	8.47	8.36	8.32	8.41	8.37	8.38	8.46
Average	8.86	8.78	8.78	8.81	8.80	8.84	8.85

these bits and use the inverse wavelet transform to reconstruct an approximation of the original image. Other specific forms of this general compression strategy can be found in [8,11–14]. In this section, we discuss two key aspects of the strategy: the order in which the bits are encoded and the estimation of probabilities-of-zero of the bits.

Following the wavelet transform, the image subbands are partitioned into segments for error-containment purposes, as described in Section V. The mean value of each segment of the LL subband is computed and subtracted from each pixel in the subband segment. (These mean values are stored in the headers of the corresponding segments so that the original values can be reconstructed.) After this step, the LL subband will in general contain both positive and negative values, even if the original pixel values were all positive.

ICER next converts all pixels in the transformed image to sign-magnitude form, i.e., each pixel is stored as a single sign bit along with several magnitude bits.[7] Conceptually, a subband can then be regarded as containing several *subband bit planes*: the most-significant magnitude bits of the pixels in the subband collectively form a single bit plane, the next most-significant magnitude bits of the pixels form another bit plane, and so on. Subband bit planes are further divided into error-containment segments using the partitioning of the subbands described in Section V.

ICER losslessly compresses the bit planes of a subband, starting with the most-significant bit plane and working toward the least significant. Compression of a subband bit plane proceeds one error-containment segment at a time, and the bits within a segment are compressed in raster scan order. The sign bits are handled differently: the sign bit of a pixel is encoded immediately after its first nonzero magnitude bit.

Bit planes from different subbands are interleaved during this coding process. Ideally, after completing compression of a subband bit plane, ICER would next compress the subband bit plane that gives the biggest improvement in some measure of image quality per compressed bit. However, there is no easy way to determine the subband that optimizes such a metric, and the gain in compression effectiveness from doing so would be rather modest in any case. Thus, ICER selects the next subband bit plane according to a simple prioritization scheme, described in Section III.A.

---

[7] It does not matter what sign bit is given to a pixel with the value 0 because the sign bit of such a pixel is never used.

Before encoding a bit, the encoder calculates an estimate of the probability that the bit is a zero. This probability-of-zero estimate relies only on previously encoded information from the same segment. We describe the mechanism for forming this estimate in Section III.A. The bit and its probability-of-zero estimate are sent to the entropy coder, which compresses the sequence of bits it receives (see Section IV).

The decompressor decodes bits within an error-containment segment in the same order they were encoded. Before decoding a bit, the decompressor therefore has available all of the information used by the encoder to produce the bit's probability-of-zero estimate, and thus can calculate the same estimate, which is essential for proper decoding. This scheme requires a bit-wise adaptable entropy coder, i.e., one with the ability to update probability estimates with each new coded bit; ICER's entropy coder, described in Section IV, has this capability.

ICER employs a technique known as *context modeling* in computing probability estimates. With this technique, a bit to be encoded first is classified into one of several *contexts* based on the values of previously encoded bits. The intent is to define contexts so that bits in the same context will have about the same probability-of-zero, and the compressor will be able to estimate this probability reasonably well from the bits it encounters in the context. We describe the specific context definitions in Section III.B and the calculation of probability estimates in Section III.C.

A well-designed context modeling scheme makes use of as much relevant contextual information as possible without producing too many contexts. Ignoring relevant contextual information (for example, by defining too few contexts) means that relevant dependencies are not exploited and results in less informative probability estimates, hurting compression effectiveness. On the other hand, if too many contexts occur, then the encoder will not be able to produce good probability estimates because it will not observe enough bits in many of the contexts.

ICER's scheme for classifying a bit of a pixel into a context is based on the previously encoded bits in pixels in the immediate neighborhood of the pixel. Some earlier methods [8,11] for compressing subband bit planes exploited similarities between different subbands, with reasonable success. However, more recent studies suggest that context models exploiting information from the immediate nine-pixel neighborhood within the given subband will gain very little from also exploiting correlations with different subbands (or, for that matter, pixels in the subband but outside this neighborhood) [13,15]. Furthermore, basing the context only on pixels in the immediate subband simplifies encoding. The particular context scheme used by ICER is derived from the scheme used by the Embedded Block Coding with Optimized Truncation (EBCOT) compressor [13] and JPEG 2000 [14], and is described in detail in Section III.B.

A. Subband Quantization and Priority Factors

Reconstructing a subband from one or more of its most-significant bit planes and the sign bits of pixels that are nonzero in at least one of these bit planes is equivalent to applying a certain scalar quantizer to each pixel of the original subband. This quantizer has quantization bins with uniform width Δ except for a central "deadzone" bin with width $2\Delta - 1$, where $\Delta = 2^b$ and b is the number of bit planes that are unavailable. Figure 6 illustrates an example of this quantizer. Including further bit planes (decreasing b) amounts to successively refining the quantizer, halving Δ for each additional bit plane.
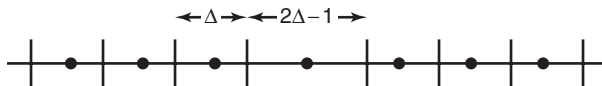

Fig. 6. Effective "deadzone" quantizer after three magnitude bits have been transmitted.

Since the wavelet transforms used by ICER produce integer outputs, we can identify ranges of integers corresponding to the quantization bins. The center deadzone bin, which corresponds to the interval $[-(\Delta - 1), \Delta - 1]$, is symmetric about the origin and, thus, we use the origin as its reconstruction point. Every other bin corresponds to an interval of the form $\pm[i\Delta, (i + 1)\Delta - 1]$, where i is a positive integer. If $\Delta = 1$, then all bit planes are available, and the pixels in the subband are reconstructed exactly. Otherwise, we use the integer $\pm((i + 1/2)\Delta - 1)$ as the reconstruction point for the bin. This point is slightly biased towards the origin since the distribution of the wavelet-transformed pixel values tends to be peaked at the origin for all but the LL subband (and the interval contains $\Delta = 2^b$ points, an even number, so the midpoint is not an integer anyway).

Subband pixel values in all but the LL subband tend to have sharply peaked and roughly symmetric distributions. For such a distribution, the deadzone quantizer induced by the bit-plane compression scheme gives better rate-distortion performance than a uniform quantizer when used for progressive compression. This is demonstrated in [16] for a Laplacian source.

Quantization of the subband data introduces distortion into the reconstructed image. Since the filters used by ICER are not unitary (in fact the linear filters that they approximate are not even orthogonal), the mean-square-error (MSE) distortion computed in the transform domain is not equal to the MSE of the reconstructed image. However, the scaled version of the transform given by $\tilde{\ell}[n] = \sqrt{2}\ell[n]$, $\tilde{h}[n] = (1/\sqrt{2})h[n]$ is approximately unitary and, thus, the weights shown in Fig. 7 can be used to determine the relative priorities of subband bit planes [7]. These weights indicate the approximate relative effect (per pixel of the subband) on the reconstructed image of root-mean-squared (RMS) distortion values in the subbands. Combining this weighting with the fact that each additional bit plane reduces the RMS distortion in a subband by roughly a factor of 2 yields relative priority weights for all subband bit planes.

For example, after three stages of wavelet decomposition, a pixel in the LL subband has a factor of 16 higher priority weight than a pixel in the level-1 HH subband. Thus, in this case the ith least-significant bit plane of the level-1 HH subband has priority equal to that of the $(i + 4)$th least-significant bit plane of the LL subband.

ICER uses this priority scheme to determine the order in which to encode subband bit planes. When multiple subband bit planes have the same priority, ICER gives precedence to those in the subbands with a higher decomposition level. When the decomposition level is also the same, the precedence depends on the type of subband, in this order: LL, HL, LH, HH.

Fig. 7. Priority weights of subbands, shown for three stages of wavelet decomposition.

B. Context Assignment

In ICER, the context of a bit in a pixel is determined by the bits already encoded in the pixel and in its eight nearest neighbors from the same segment of the subband. Thus, from the more-significant bit planes, bits from all nine of these pixels help to determine the context; from the current bit plane, bits from only four of these pixels are used (they are determined by the raster-scan encoding order within bit-plane segments). See Fig. 8.

As it compresses bit planes, ICER keeps track of a *category* for each pixel. A pixel's category summarizes information about the bits already encoded in the pixel, facilitating the capture of relevant contextual information with a small number of contexts. ICER uses four pixel categories that are based on the concept of *significance* of a pixel [11]: a pixel is not significant if the magnitude bits already encoded in the pixel are all zeros; otherwise, the pixel is significant. We label the categories as 0, 1, 2, and 3. A pixel's category is 0 if it is not yet significant; after the first '1' bit from the pixel is encoded, the pixel's category becomes 1; when the next magnitude bit from the pixel is encoded, the pixel's category becomes 2; and, finally, when one more magnitude bit from the pixel is encoded, the pixel's category becomes 3 and remains 3 permanently. Figure 9 provides an example of the categories a pixel goes through as it is encoded (here MSB and LSB denote the most-significant bit and least-significant bit, respectively).

Bits to be encoded that are likely to be compressible are classified into one of 17 contexts; as mentioned above, this classification scheme is derived from that of EBCOT [13] and JPEG 2000 [14]. Contexts 0 through 8 are used for bits of pixels that are not yet significant (that is, pixels in category 0). Contexts 9 and 10 are used for bits of pixels that have just become significant (pixels in category 1). Context 11 is used for bits of pixels in category 2. Contexts 12 through 16 are used for sign bits.

Bits of pixels in category 3 are empirically nearly incompressible; that is, estimates of these bits' probabilities-of-zero tend to be very close to $1/2$. Therefore, these bits are left uncoded in the compressor's output.[8]

The context of a bit is determined using the category of the pixel containing the bit and the significance and signs of the 8 adjacent pixels. If not all of the 8 adjacent pixels are available because the pixel being encoded is at the edge of its subband segment, the missing pixels are treated as being not yet significant.
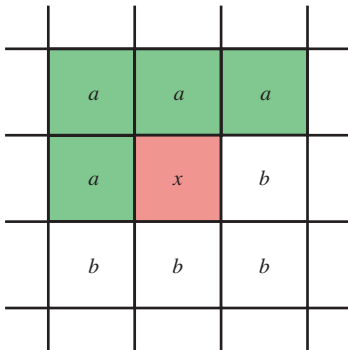

Fig. 8. When encoding a bit of the pixel "x," the context is determined by bits already encoded in the nine pixels shown. Magnitude bits from the current and more significant bit planes are available from the "a" pixels, while only magnitude bits from the more significant bit planes are available from pixel "x" and the "b" pixels.

---

[8] More precisely, they are sent directly to the "uncoded" bin of the interleaved entropy coder (see Section IV), implicitly assuming a probability-of-zero of $1/2$.
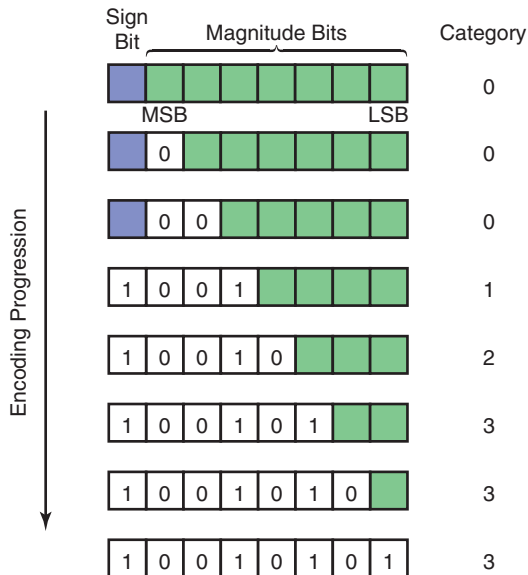
Fig. 9. Example of the progression of categories of a pixel as its magnitude bits and sign are encoded. The pixel shown has value −21, and an 8-bit representation is assumed (7 magnitude bits preceded by a sign bit). Bits not yet encoded are shaded. Between each state shown, a magnitude bit is encoded as part of the encoding of a subband bit plane. This pixel's sign bit is encoded with the third most-significant bit plane.

We first describe the contexts for bits of pixels in category 0. If the subband being encoded is *not* an HL subband, then let h be the number of horizontally adjacent pixels that are significant (0, 1, or 2), v be the number of vertically adjacent pixels that are significant (0, 1, or 2), and d be the number of diagonally adjacent pixels that are significant (0–4). For an HL subband, the roles of h and v are reversed, effectively transposing the context template. Given h, v, and d, the context is assigned according to Table 6 if the subband is not an HH subband; otherwise, the context is assigned according to Table 7.

A bit of a pixel in category 1 is assigned context 9 if none of the horizontally or vertically adjacent pixels is significant; otherwise, it is assigned context 10. A bit of a pixel in category 2 is assigned context 11 regardless of the categories of adjacent pixels.

Sign bits are not encoded directly; rather, the context modeler first predicts the sign bit and then encodes an "agreement" bit that is the exclusive-or of the sign bit and its predicted value. The agreement-bit statistics associated with a sign context are used to estimate a probability-of-zero for future agreement bits in the context in exactly the same manner as magnitude bits in magnitude contexts. The prediction of

Table 6. Contexts for bits of pixels in category 0 in LL, LH, and HL subbands, as a function of h, v, and d.

d	$h = 0$			$h = 1$		$h = 2$
	$v = 0$	$v = 1$	$v = 2$	$v = 0$	$v > 0$	
$d = 0$	0	3	4	5	7	8
$d = 1$	1	3	4	6	7	8
$d \geq 2$	2	3	4	7	7	8

17

Table 7. Contexts for bits of pixels in category 0 in HH subbands, as a function of $h + v$ and d.

d	$h + v = 0$	$h + v = 1$	$h + v \geq 2$
$d = 0$	0	1	2
$d = 1$	3	4	5
$d = 2$	6	7	7
$d \geq 3$	8	8	8

sign bits allows the number of contexts to be reduced, as suggested by the following reasoning: Suppose A and B are pixels with as-yet-unencoded signs. If the pixels in the neighborhood of A are the negatives of the corresponding pixels in the neighborhood of B, then by symmetry we should expect the probability that A is positive is about the same as the probability that B is negative. Therefore, the agreement bits for both pixels' signs can be sensibly encoded with the same context, even if the sign bits themselves could not be.

ICER uses the two horizontally adjacent and the two vertically adjacent pixels to determine both the sign estimate and the context. If the subband is not an HL subband, let h_1 and h_2 represent the signs and significances of the two horizontally adjacent pixels, taking on the values 1, -1, and 0 for pixels that are positive, negative, and not significant, respectively. Similarly, let v_1 and v_2 represent the signs and significances of the two vertically adjacent pixels. For the HL subband, the roles of the h's and v's are again reversed. Otherwise, the different types of subbands are treated the same with respect to encoding sign bits. Table 8 lists the sign estimate and the context as a function of $h_1 + h_2$ and $v_1 + v_2$.

C. Probability Estimation

For each context, ICER maintains nominal counts of the number of zero bits and the total number of bits that have occurred in the context; the ratio of these counts represents the probability-of-zero estimate for the context. For each bit to be encoded, the entropy coder receives the probability-of-zero estimate in the form of the ratio of these counts.

Each context's counts are initialized to values corresponding to a probability-of-zero of 1/2. Each bit encountered in a context increments the total count, and increments the count of zeros if the bit is a 0. (Of course, these increments occur after the bit is encoded.) When the total count reaches a specified value, both counts are rescaled by dividing by 2 (when necessary, the count of zeros is rounded in the direction that makes the probability-of-zero estimate closer to 1/2). The rescaling has the effect of producing probability-of-zero estimates that give more weight to recent bits, accommodating to some degree context statistics that change as the compression proceeds.

In our MER implementation, the initial counts of zeros are set to 2, the initial total counts are set to 4, and rescaling is triggered when the total count reaches 500.

Table 8. Sign bit predictions and sign contexts.

$v_1 + v_2$	$h_1 + h_2 < 0$	$h_1 + h_2 = 0$	$h_1 + h_2 > 0$
$v_1 + v_2 < 0$	$-$, 16	$+$, 13	$+$, 14
$v_1 + v_2 = 0$	$-$, 15	$+$, 12	$+$, 15
$v_1 + v_2 > 0$	$-$, 14	$-$, 13	$+$, 16

D. Coding Differences between ICER, EBCOT, and JPEG 2000

ICER's bit-plane coding has many similarities to bit-plane coding in EBCOT [13] and JPEG 2000 [14]. Most notably, the context scheme used by ICER is derived from that of EBCOT (whose context scheme is nearly the same as that of JPEG 2000). However, there are also many differences in how the three compressors perform bit-plane coding. We mention some of the bigger differences here. We also point out differences related to entropy coding.

EBCOT and JPEG 2000 organize subbands into smaller units in a way that is somewhat different from ICER's organization of subbands into error-containment segments. JPEG 2000 uses a more complicated order for encoding subband bit planes than the simple raster-scan order used in ICER. During encoding, EBCOT transposes HL subbands, while ICER and JPEG 2000 transpose only the context template of such subbands (see Section III.B). ICER makes just one pass through each subband bit plane; JPEG 2000 makes three passes, each pass encoding a different subsets of bits; and EBCOT makes four such passes, one of which is in reverse raster-scan order.

There are several differences in the way the three compressors perform probability estimation and entropy coding. JPEG 2000 uses an approximate arithmetic coder (the MQ coder from the JBIG2 standard [17]) for entropy coding; EBCOT uses more standard arithmetic coding; and ICER uses an interleaved entropy coder (see Section IV). Probability estimation in JPEG 2000 is performed using a state table incorporated in the arithmetic coding process. EBCOT and JPEG 2000 group 4 bits together (in a form of run-length coding) prior to entropy coding under certain circumstances; ICER does not do anything similar to this. ICER regards certain bits as incompressible and leaves them uncoded (see Section III.B); this is similar to the "lazy coding" option in JPEG 2000, but EBCOT has nothing analogous. Finally, JPEG 2000 uses skewed initial statistics for some contexts so that the initial probability-of-zero estimates for bits in these contexts are not $1/2$; this can slightly improve compression effectiveness on typical images. EBCOT and ICER use unskewed initial statistics for all contexts (see Section III.C).

IV. Entropy Coding

In this section, we describe the entropy coder used by ICER to compress the magnitude and sign bits of the pixels in the wavelet-transformed image.

A. Adaptable Entropy Coding

In a sequence of source bits $b_1, b_2, \cdots$ from the subbands, for each bit b_i the context modeler described in Section III produces an estimate p_i of the probability that b_i equals zero. The entropy coder uses these estimates to produce an encoded (and hopefully compressed) bitstream from which the original bit sequence can be reconstructed.

ICER uses a bit-wise adaptable entropy coder. Thus, the probability estimate p_i can depend on the values of previous bits and change with each new bit, allowing the context modeler to quickly adapt to changing statistics and make better use of the immediate context in which a bit appears. Accommodating a probability estimate that varies from bit to bit in this way is tricky because the decompressor needs to construct the same estimates as the compressor. Thus, the decompressor must determine the values of the first $i - 1$ bits before it can decode the ith bit.

Although adaptable binary entropy coding usually is performed using arithmetic coding [18] (or with a low-complexity approximation to arithmetic coding such as that described in [19]), we have chosen to use a lesser-known technique called *interleaved entropy coding* [20–22]. Given perfect probability estimates, practical implementations of both arithmetic coding and interleaved entropy coding can compress stochastic bit sequences to within 1 percent of the theoretical limit, but interleaved entropy coding has some speed advantages [21].

Interleaved entropy coding was first suggested in [23] and has also appeared in [20,24]. In [21], we introduce *recursive* interleaved entropy coding, a generalization of the technique. References [21,22] contain a thorough description of recursive and non-recursive interleaved entropy coding. In the trade-off between encoding speed and compression effectiveness, non-recursive coders generally outperform recursive coders. Thus, we have used a non-recursive coder in ICER, and in the remainder of this article the discussion of interleaved entropy coding will implicitly refer to the non-recursive type. Our software implementation of the coder in ICER has particularly low complexity, and so is well-suited for space applications and other applications where encoding speed can be of critical importance.

In the remainder of this section, we give a brief overview of interleaved entropy coding and specify the particular coder design used in ICER.

B. Variable-to-Variable-Length Codes

An interleaved entropy coder combines several component source codes. Each component code is a variable-to-variable-length binary source code, which is a mapping from a set of input codewords to a set of output codewords. *Variable-to-variable length* means that in general neither the input codewords nor the output codewords all have the same length. An encoder for such a code parses a sequence of input bits into input codewords. The encoder output consists of the concatenation of the corresponding output codewords. Decoding is accomplished using the same procedure as encoding, with the roles of input and output codeword sets reversed.

An example of a variable-to-variable-length binary source code is given in Table 9 (this code is the Golomb code [25] with parameter $m = 5$; see below for a description of the family of Golomb codes). With this code, the binary sequence 0100000001100001 is parsed as $01, 00000, 001, 1, 00001$, resulting in the encoded output sequence $001, 1, 010, 000, 0111$.

For every component code, the input and output codeword sets are each *prefix-free*, which means that no codeword is a prefix of another codeword in the set, and *exhaustive*, which means that no codeword can be added to the set without violating the prefix-free condition. These properties ensure that a sequence of bits can be uniquely parsed into codewords and that we recognize a complete codeword as soon as it is formed, i.e., without looking ahead in the sequence.

Several of the component codes in ICER's entropy coder are Golomb codes [25,26], which are efficient for encoding long runs of bits having nearly the same probability-of-zero, especially when that probability-of-zero is near 1. We denote particular Golomb codes with the notation $\mathcal{G}_m$, where $m \geq 1$. The code $\mathcal{G}_m$ has $m + 1$ input codewords: $1, 01, 001, \cdots, 0^{m-1}1$, and 0^m. (We use the notation 0^i to denote a run of i zeros.) Input codeword 0^m maps to the output codeword consisting of a single 1. To describe the output codewords for the other input codewords, we let $\ell = \lceil \log_2 m \rceil$ and $i = 2^\ell - m$. The output

Table 9. The Golomb code with parameter $m = 5$.

Input codeword	Output codeword
00000	1
00001	0111
0001	0110
001	010
01	001
1	000

codeword corresponding to input codeword 0^k1 is the ℓ-bit binary representation of integer k when $k < i$, and the $(\ell + 1)$-bit representation of the integer $k + i$ otherwise.

C. Interleaved Entropy Coding

An interleaved entropy coder compresses a binary source with a bit-wise adaptive probability estimate by interleaving the output of several different variable-to-variable-length binary source codes that each encode groups of bits with similar probability estimates. Much variation is possible in the choice and number of component codes, yielding coder designs of varying complexity and compression efficiency.

Without loss of generality, we may assume that $p_i \geq 1/2$ for each index i. If this is not the case for some p_i, we simply invert bit b_i before encoding to make it so; this inversion clearly can be duplicated in the decoder. Thus, we are concerned with the probability region $[1/2, 1]$. We partition this region into several narrow intervals or *bins*, and for each bin we define a component code designed to effectively compress sequences of bits having probabilities-of-zero lying in this interval. Bit b_i is encoded (along with other source bits) using the source code corresponding to the interval containing p_i.

For example, one of the bins of ICER's entropy coder corresponds to a probability interval approximately equal to $(0.85, 0.88)$ and uses the Golomb code $\mathcal{G}_5$ shown in Table 9. To give an indication of how well this code performs in this interval, we can evaluate the compression effectiveness of this code when applied to an independent and identically distributed (IID) binary source that produces zeros with fixed probability p in this interval. A short calculation shows that the average input codeword length equals $1 + p + p^2 + p^3 + p^4$ and the average number of output bits per codeword is $3 + p^3 - 3p^5$ for this source. Thus, the average rate is

$$r(p) = \frac{3 + p^3 - 3p^5}{1 + p + p^2 + p^3 + p^4} \quad \text{(bits/source bit)}$$

and the redundancy is $r(p) - \mathcal{H}_2(p)$ where

$$\mathcal{H}_2(p) = -p \log_2 p - (1 - p) \log_2(1 - p)$$

is the binary entropy function, the theoretical limit of compression achievable for this source. Figure 10 shows that the redundancy for this example is small when p is in the appropriate range.

The output of any interleaved entropy coder contains some redundancy because, for example, source bits with slightly different estimated probabilities-of-zero are treated the same; that is, the bins' intervals have positive widths. However, by using an increasing number of increasingly large variable-to-variable-length codes, one can produce a coder design with redundancy as small as desired (asymptotically as the input sequence length becomes long), given a stochastic source with perfect probability estimates [21].

We now give brief descriptions of the encoding and decoding procedures. Refer to [21,22] for a more complete treatment.

1. Encoding. The encoder groups bits from the same bin together to form input codewords. To ensure that decoding is possible, the output codewords produced by the different component codes must be interleaved in the proper order. The encoder accomplishes this by maintaining a list of input codewords and partially formed input codewords in a circular buffer. In the MER implementation of ICER, the circular buffer has a capacity of 2048 words.

When source bit b_i arrives, it is assigned to the bin whose probability interval contains the bit's probability-of-zero estimate p_i. The encoder then checks whether the list includes a partial input codeword
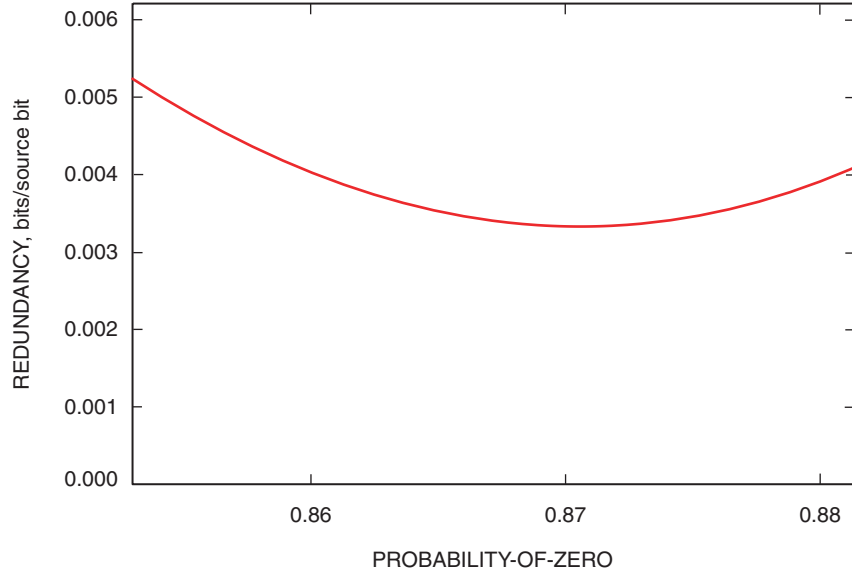
Fig. 10. Redundancy for the Golomb code $\mathcal{G}_5$ used on an IID binary source with fixed probability-of-zero. ICER uses this code for the probability-of-zero range covered by this graph.

for this bin. If so, the bit is appended to this partial codeword. Otherwise, the bit begins a new word at the end of the list.

When the word at the beginning of the list is a complete input codeword for the corresponding component code, the encoder produces the corresponding output bits and the word is removed from the list, making more room available for new words. If the new beginning position of the list contains a complete codeword, it also is processed in this manner, and so on.

If the buffer containing the word list becomes full, one or more "flush bits" are appended to the (necessarily partial) input codeword at the front of the list to form a complete codeword that is then processed in the normal manner. Flushing of a partial input codeword is accomplished by producing the shortest output codeword that is consistent with the bits already in the partial codeword; in practice, the appropriate output codewords corresponding to partial input codewords are tabulated in advance. Flush bits also are used to complete all partial codewords remaining in the list of words once the input bit sequence is exhausted.

2. Decoding. The decoder is somewhat simpler than the encoder. The decoder keeps track of a partial input codeword for each bin of the coder. In the decoder, each such word is a *suffix* of an input codeword for the bin. Codewords are reconstructed in the decoder in the same order that they appeared in the encoder's list.

Source bits are decoded in order. To decode b_i, the decoder calculates the associated probability estimate p_i, which in general may be based on the values of preceding source bits just decoded. Given p_i, the decoder uses the same procedure as the encoder to determine the bin to which bit b_i was assigned by the encoder. If the decoder has a partial codeword for this bin, the first remaining bit of this partial codeword is removed; this bit becomes the decoded bit b_i. Otherwise, the decoder must reconstruct an input codeword of the component code for the bin; this is accomplished by parsing an output codeword from the next available bits in the encoded bitstream and determining the corresponding input codeword. The first bit of this codeword becomes the decoded bit, and the decoder remembers the remaining codeword suffix.

For proper decoding, the decoder must identify and remove flush bits so that they are not mistaken for source bits. Fortunately, this is straightforward. While decoding, the decoder keeps count of the number of codewords reconstructed. Each time a codeword is reconstructed, the decoder stores this count along with the codeword suffix. When the difference between the current count and the value associated with a given codeword suffix exceeds the size of the encoder's buffer, then all of the remaining bits in this suffix must be flush bits and are discarded. Identification of flush bits can be accomplished quite efficiently; in [22] we describe several approaches in detail.

D. ICER's Interleaved Entropy Coder

In this section, we specify the particular interleaved entropy coder design used by ICER.

We first describe a shorthand notation, similar to that introduced in [21], that we use to specify some of the component codes. As an alternative to a code table, a variable-to-variable-length source code can be specified by a tree that provides a map for decoding. For example, Fig. 11 shows a decoding tree for the Golomb code $\mathcal{G}_5$ of Table 9. Each input codeword is assigned to a leaf in the tree, and the branch labels, each a zero or one, indicate the output bits. The output codeword assigned to an input codeword is the sequence of branch labels from the root to the leaf for that input codeword. Thus, to decode, we start at the root, traversing a branch for each encoded bit read, until we reach the input codeword at the leaf. (Similarly, one could construct an encoder-oriented tree.)

We use a shorthand description of the decoding tree as a compact specification of a component code. For each terminal node, we write the corresponding input codeword. A non-terminal node is represented by an ordered pair containing the representations of the child nodes, with the node associated with a zero output bit listed first. Thus, using this shorthand, the Golomb code $\mathcal{G}_5$ shown in Fig. 11 and Table 9 also can be represented as

$$\Big(\big((1, 01), \big(001, (0^3 1, 0^4 1) \big) \big), 0^5 \Big)$$

An interleaved entropy coder design is specified by giving the probability interval and the component source code used for each bin. The bins are indexed starting from 1, with lower indices assigned to probability intervals closer to $1/2$. The intervals may be specified by probability cutoffs z_j so that bin j corresponds to probability interval $[z_{j-1}, z_j)$, where z_0 is taken to be $1/2$.

Table 10 specifies ICER's entropy coder design, which has 17 bins. To simplify the comparison between probability estimates and cutoffs, we use cutoffs that are rational numbers with denominator 2^{16}. Bits in
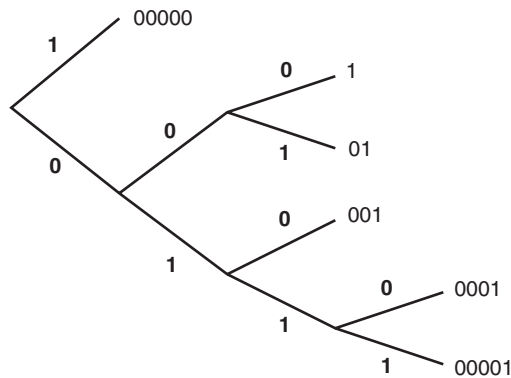

Fig. 11. A decoding tree for the Golomb code $\mathcal{G}_5$. Output bits are shown in boldface. The input codewords are shown at terminal nodes of the trees.

Table 10. The interleaved entropy coder design used by ICER.

Bin index j	Probability cutoff z_j	Code
1	35298/65536	$(0,1)$ (uncoded)
2	37345/65536	$(((((0^41,1^4),0^31),001),10),(01,(110,(0^5,1^30))))$
3	40503/65536	$(((001,((1101,0^311),1^3)),10),(01,(0^4,(1100,0^310))))$
4	43591/65536	$((0^3,01),(10,(001,11)))$
5	47480/65536	$(((010,(10^4,110)),((101,011),((10^31,1^3),1001))),00)$
6	50133/65536	$((0^5,1),((0^31,001),(010,(0^41,011))))$
7	53645/65536	$(0^3,((001,010),(100,(11,(011,101)))))$
8	55902/65536	$(0^4,((001,01),(10,(0^310,(0^311,11)))))$
9	57755/65536	$\mathcal{G}_5$
10	58894/65536	$\mathcal{G}_6$
11	60437/65536	$\mathcal{G}_7$
12	62267/65536	$\mathcal{G}_{11}$
13	63613/65536	$\mathcal{G}_{17}$
14	64557/65536	$\mathcal{G}_{31}$
15	65134/65536	$\mathcal{G}_{70}$
16	65392/65536	$\mathcal{G}_{200}$
17	65536/65536	$\mathcal{G}_{512}$

the first bin, whose interval contains probability $1/2$, are incompressible (or nearly so), and are unchanged by the coding process. For this bin, each source bit forms a complete input codeword, and each output codeword equals the input codeword. The component codes for bins 2 through 8 are specified using our shorthand notation for the code's tree. For bins 9 through 17, corresponding to probability-of-zero estimates larger than $55902/65536 \approx 0.853$, ICER uses the Golomb codes indicated in the table.

As an indication of the effectiveness of ICER's entropy coder, Fig. 12 shows the asymptotic redundancy obtained when the coder is used to compress an IID source with fixed but known probability-of-zero. The graph shows that, for a stochastic source with perfect probability estimates, the redundancy contribution from ICER's entropy coder is quite small. Thus, to improve ICER's compression effectiveness, we would be more inclined to invest our efforts in an area such as improved context modeling rather than trying to reduce the redundancy of ICER's entropy coder.

V. Error Containment

Data-compression methods are more effective when they exploit dependencies between the data being compressed and previously compressed data in the same image or data set. However, a drawback of exploiting such dependencies, especially over large data sets, is that if a portion of compressed data is lost, then decoding of subsequent dependent portions is impossible.

Thus, to mitigate the impact of data losses that occur on the deep-space communications channel, a data compressor must incorporate effective error-containment techniques. Without error containment, even a small loss of data can render useless large portions of compressed data. Error containment can be achieved by dividing the data set into segments that are compressed independently, so that when data from one segment are lost or corrupted, reconstruction of the other segments is unaffected.

In this section, we describe data-loss mechanisms of the deep-space channel and ICER's particular error-containment strategy.
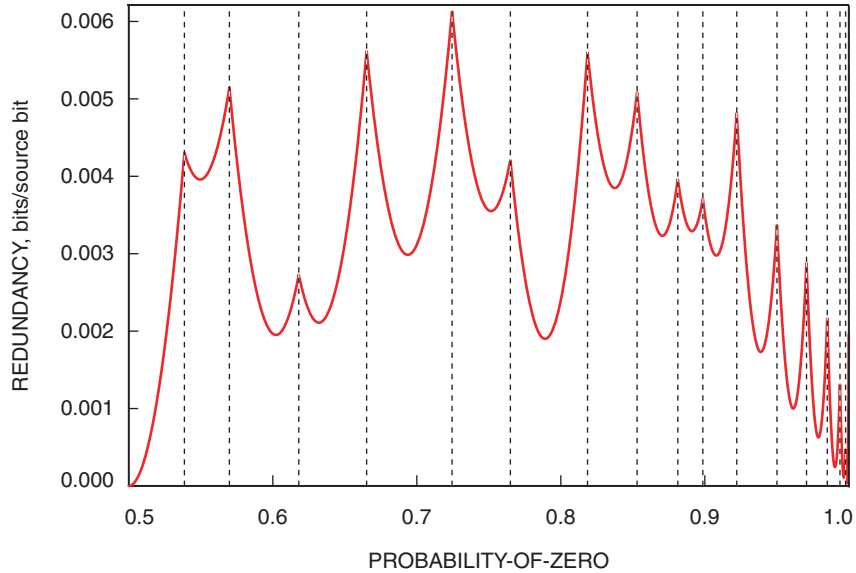
Fig. 12. Redundancy as a function of probability-of-zero when ICER's interleaved entropy coder is used to compress an IID binary source with a known fixed probability-of-zero. The dashed vertical lines mark the boundaries between bins.

A. Data Loss on the Deep-Space Channel

The Consultative Committee for Space Data Systems (CCSDS) packet telemetry standard [27] defines the protocol used for the transmission of spacecraft instrument data over the deep-space channel. Under this standard, an image or other data set from a spacecraft instrument is transmitted using one or more *packets*. A packet is a block of data with length that can vary between successive packets, ranging from 7 to 65,542 bytes, including the packet header. Packetized data are transmitted via *frames*, which are fixed-length data blocks. The size of a frame, including frame header and control information, can range up to 2048 bytes, but it is fixed during a given mission phase. Because packet lengths are variable but frame lengths are fixed, packet boundaries usually do not coincide with frame boundaries, as illustrated in Fig. 13.

Data in a frame typically are protected from channel errors by error-correcting codes. Even when the channel errors exceed the correction capability of the error-correcting code, the presence of errors nearly always is detected by the error-correcting code or by a separate error-detecting code. Frames for which uncorrectable errors are detected are marked as undecodable and typically are deleted. Deleted undecodable whole frames are the principal type of data loss that affects compressed data sets.

There generally would be little to gain from attempting to use compressed data from a frame marked as undecodable. When errors are present in a frame, the bits of the subband pixels already decoded before the first bit error will remain intact, but all subsequent decoded bits in the segment usually will
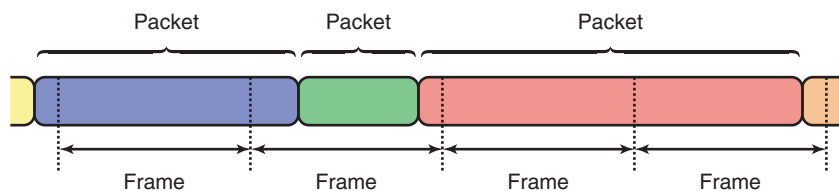

Fig. 13. Packet boundaries usually are not aligned with frame boundaries.

be completely corrupted; a single bit error is often just as disruptive as many bit errors. Furthermore, compressed data usually are protected by powerful, long-blocklength error-correcting codes, which are the types of codes most likely to yield substantial fractions of bit errors throughout those frames that are undecodable. Thus, frames with detected errors would be essentially unusable even if they were not deleted by the frame processor.

If an erroneous frame escapes detection, the decompressor will blindly use the frame data as if they were reliable, whereas in the case of detected erroneous frames, the decompressor can base its reconstruction on incomplete, but not misleading, data. Fortunately, it is extremely rare for an erroneous frame to go undetected. For frames coded by the CCSDS Reed–Solomon code [28], fewer than 1 in 40,000 erroneous frames can escape detection [29]. All frames not employing the Reed–Solomon code use a cyclic redundancy check (CRC) error-detecting code, which has an undetected frame-error rate of less than 1 in 32,000 [30].

To summarize, coding and framing considerations lead us to model the relevant data losses as losses of whole frames. A frame loss results in the loss of one or more entire or partial packets, depending on the packet lengths and the alignment between packet and frame boundaries (refer to Fig. 13). When a frame loss eliminates only a portion of a packet, intact data in the packet preceding the loss could be used by the decompressor to reconstruct source data. However, keeping track of partial packets increases the complexity of ground operations, and the usual practice is to discard incomplete packets. Thus, the effective consequence of a frame loss is generally the loss of one or more entire packets.

Since the loss of a single frame may affect more than one packet, for error-containment effectiveness it is important that ICER's compressed output be arranged so that packets containing data from the same segment are contiguous, as described in Section VI.B. This arrangement reduces the chance that a single frame loss affects more than one segment.

B. Error Containment in ICER

To achieve error containment, ICER automatically partitions the image data into a user-specified number of *segments*. Each segment is compressed independently of the others so that the loss of data from one segment does not affect the ability to reconstruct another segment.

To ensure that the segments can be decompressed independently of each other, each compressed segment begins with a header containing information such as the segment index, image dimensions, and ICER parameters. Because each segment is compressed independently, ICER maintains separate context modeler and entropy coder data for each segment. When organizing compressed data into packets for transmission, multiple packets may be used for a single compressed segment, but no packet should contain data from more than one segment.

Conceptually, segmentation occurs after the wavelet decomposition. Each pixel of the transformed image is assigned to a segment. Although segments are defined in the transform domain, each approximately corresponds to a rectangular region of the original image. If s segments are desired, first the LL subband is partitioned into s rectangular segments, and then this partition is mapped to the other subbands. Thus, pixels in different subbands corresponding to the same spatial location will belong to the same segment. As a simple example, we illustrate in Fig. 14 the regions of a transformed image corresponding to a partition into two segments. The remaining problem of how to partition the LL subband into rectangles is addressed in Section V.D.

A consequence of defining segments in the wavelet transform domain is that segment boundaries are not sharply defined in the image domain. To reconstruct pixels near the boundaries between segments, the inverse wavelet transform combines data from adjacent segments. Consequently, the effect of data loss in one segment can appear to "bleed" slightly into adjacent segments in the reconstructed image; i.e., a few pixels near the borders of that segment may appear blurred.
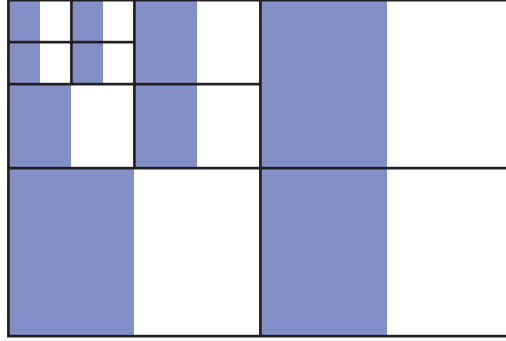
Fig. 14. A partition of a wavelet-transformed image into two segments, shown here for three stages of wavelet decomposition. The shaded pixels all belong to the left segment.

Defining image segments in the wavelet transform domain has some advantages over the simpler alternative of partitioning the image directly and applying a wavelet decomposition separately to each segment (i.e., dividing the original image into smaller images that are compressed independently). With lossy compression under this simpler alternative, the boundaries between segments would tend to be noticeable in the reconstructed image even when no compressed data are lost, as illustrated in Fig. 15(a). This is, in fact, a larger-scale version of the "blocking" artifact that can occur in JPEG-compressed images (see Fig. 2(c), for example). By segmenting the image in the transform domain, we can virtually guarantee that such artifacts will not occur. Figure 15(b) illustrates the elimination of these artifacts.

By applying the wavelet transform to the entire image at once, we also achieve better decorrelation and reduce inefficiencies associated with the wavelet transform at image boundaries, thus improving compression effectiveness.

Because ICER compresses each segment of a subband bit plane before moving on to the next subband bit plane, each segment will be encoded to nearly the same fidelity. The lengths of compressed segments will vary somewhat, in large part because segments that are easier to compress will use fewer bits. For example, a segment consisting mostly of smooth sky likely would use fewer bits than a similarly sized segment containing highly textured terrain and rocks.

Since each segment is compressed progressively, some error containment automatically occurs within segments as well: when a packet loss occurs, any previously received packets for the affected segment will allow a lower-fidelity reconstruction of that segment. Although the decompressor is unable to make use of data in a segment that follows a loss, a packet loss does not eliminate an entire segment unless the packet lost is the initial packet for the segment. Figure 16 illustrates an image reconstructed following packet losses affecting three segments.
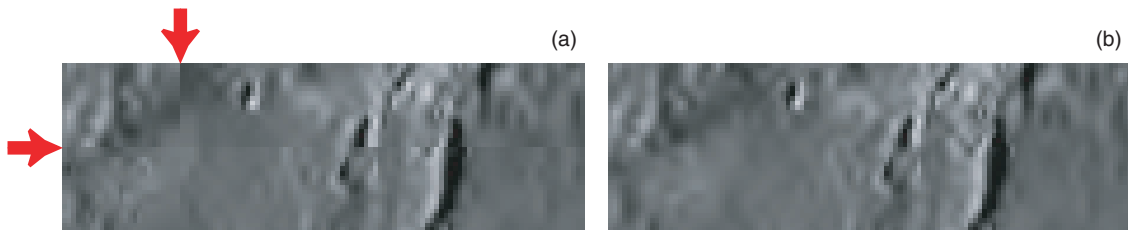

Fig. 15. Image details that illustrate (a) artifacts at segment borders that would arise if the wavelet transform were separately applied to each segment and (b) elimination of such artifacts in the ICER-compressed image.
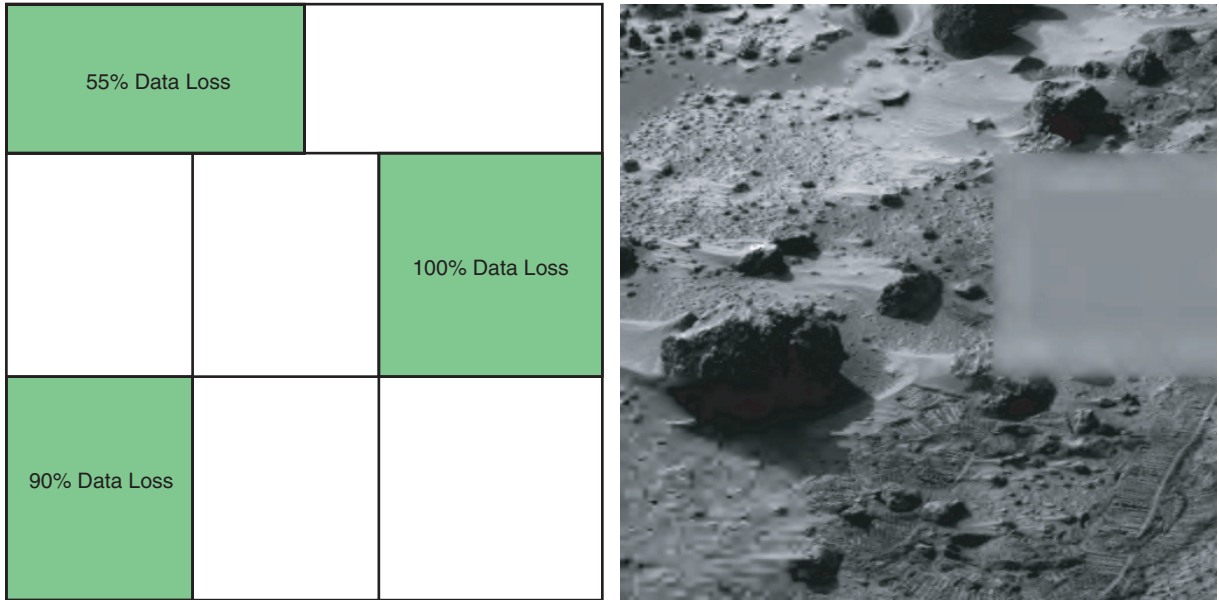
Fig. 16. Example of error containment in an image compressed to 1 bit/pixel, with simulated packet losses affecting three of eight image segments.

C. Choosing the Number of Segments

ICER provides flexibility in choosing the number of segments, allowing a user to trade compression effectiveness against packet loss protection, thereby accommodating different data loss rates. In general, we would tend to use a larger number of segments for larger images, larger numbers of compressed bytes, and less reliable channels.

Dividing an image into a larger number of segments tends to result in increased protection from data loss because a loss within a single segment affects a smaller portion of the image. However, there is a limit to the added protection provided by increasing the number of segments: as the segments decrease in size, the number of compressed bytes per segment becomes small, and it becomes more likely that a single frame loss will affect multiple segments. Moreover, compression effectiveness suffers when image segments become sufficiently small, in part because smaller segments have less data over which dependencies can be exploited. Thus, as the number of segments increases, at some point the marginal error-containment benefit becomes tiny, and the compression effectiveness cost becomes significant. Note that, for a given number of segments, the former is determined primarily by the number of output bytes produced, and the latter is determined primarily by the image size.

The MER implementation of ICER limits the number of segments to 32 or fewer and imposes tighter limits when the image is small and the number of stages of wavelet decomposition is large.[9] Generally, however, a user has considerable flexibility in choosing the number of segments, especially for large images.

If packet losses are rare or when compressing a small image, one might reasonably set the number of segments to 1 so that the entire image is treated as a single segment. However, some amount of segmentation may slightly improve compression effectiveness, especially on large images. Many images are most effectively compressed using four to six segments because segmentation can coincidentally separate disparate regions of the image into different segments (e.g., some segments of an image may consist mostly of sky and others mostly of soil), allowing the context modeler to develop distinct sets of statistics for these regions.

---

[9] A. Kiely and M. Klimesh, op. cit.

D. The Partitioning Algorithm

As described in Section V.B, ICER determines segment boundaries for the LL subband first, and then maps them to the other subbands. We now describe ICER's algorithm for partitioning the LL subband. Essentially, this algorithm partitions a rectangle with integer side lengths into smaller rectangles that also have integer side lengths.

The partitioning algorithm has the following desirable properties:

(1) The algorithm is fairly simple and requires no floating-point operations.

(2) The segments tend to be nearly square. A square region has a smaller perimeter than an elongated region of the same area, so its pixels have more neighbors with which correlations can be exploited. Thus, more effective compression is possible with square segments.

(3) The segments tend to have nearly equal areas. This makes it somewhat more likely that segments will have similar compressed lengths, which means that data losses are likely to affect smaller regions of the image.

(4) The algorithm will give a valid result (notably, no zero-width or zero-height segments) as long as the area of the rectangle is at least as large as the desired number of segments.

This last property implies that before segmenting an image we need check only that the number of pixels in the lowest subband of an image is at least as large as the number of segments. (In fact, a stronger condition is enforced in the MER implementation of ICER.[10]) In Section V.E, we show that Property (4) holds, and we provide support for our claim that Properties (2) and (3) hold.

In the remainder of this section, w and h refer to the width and height of the LL subband. As noted in Section II.B, if the image has width W and height H, then $w = \lceil W/2^D \rceil$ and $h = \lceil H/2^D \rceil$, where D is the number of wavelet decompositions.

The inputs to the partitioning algorithm are w, h, and the number of segments s, all of which must be positive integers. The condition that the area of the rectangle is at least as large as the number of segments thus can be written

$$s \leq wh \tag{9}$$

The algorithm produces a partition, consisting of s smaller rectangles, that is conveniently described by several parameters that we will regard as the output of the algorithm.

The segments are arranged in r rows. A "top region" of the rectangle contains segments arranged in c columns. A possible "bottom region" contains segments arranged in $c + 1$ columns. The height of the top region is h_t, and it contains r_t rows of segments. The first r_{t0} rows of segments in the top region have height y_t; the remaining rows (if any) have height $y_t + 1$. The first c_{t0} columns in the top region have width x_t; any remaining rows have width $x_t + 1$. Similarly, the first r_{b0} rows in the bottom region have height y_b and the remainder have height $y_b + 1$, and the first c_{b0} columns in the bottom region have width x_b and the remainder have width $x_b + 1$. Figure 17 gives an example showing a partition and the parameters describing it.

Indices are assigned to segments in raster scan order, as shown in Fig. 17. The decompressor applies the same segmentation algorithm to compute the segment boundaries from the image dimensions, the
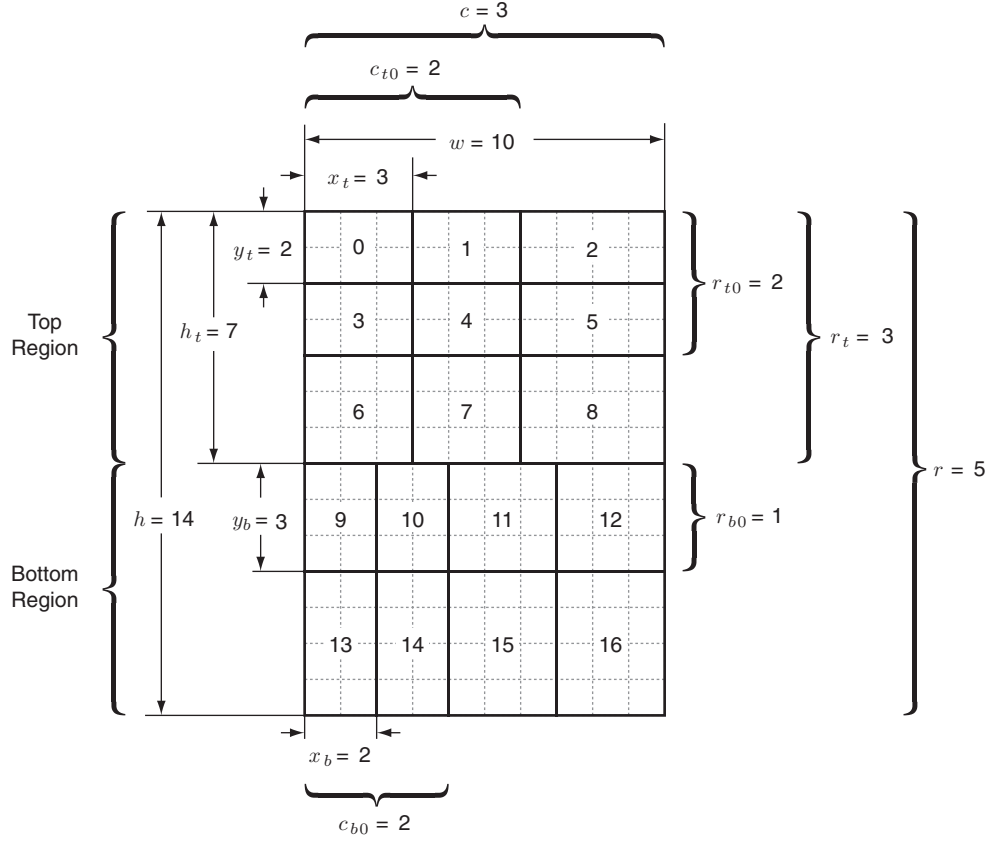
---

[10] Ibid.

Fig. 17. The partition, and parameters describing it, when $w = 10$, $h = 14$, **and** $s = 17$. **Each segment is labeled with an index number. A fairly large number of segments is shown for illustrative purposes.**

number of stages of wavelet decomposition, and the total number of segments. These values are encoded in a header at the beginning of each compressed segment; thus, boundaries of individual segments are not explicitly encoded.

The partitioning algorithm proceeds as follows.

First r is computed. If $h > (s-1)w$, then $r = s$. Otherwise, r is the unique positive integer satisfying

$$(r-1)rw < hs \le (r+1)rw \tag{10}$$

In practice, r is computed with the following C code:

```
for (r=1; r<s && (r+1)*r*w < h*s; r++)
    ;
```

The other parameters are computed as follows:

$$c = \left\lfloor \frac{s}{r} \right\rfloor \tag{11}$$

$$r_t = (c+1)r - s \tag{12}$$

30

$$h_t = \max\left(r_t, \left\lfloor \frac{hcr_t}{s} + \frac{1}{2} \right\rfloor\right) \tag{13}$$

$$x_t = \left\lfloor \frac{w}{c} \right\rfloor \tag{14}$$

$$c_{t0} = (x_t + 1)c - w \tag{15}$$

$$y_t = \left\lfloor \frac{h_t}{r_t} \right\rfloor \tag{16}$$

$$r_{t0} = (y_t + 1)r_t - h_t \tag{17}$$

If $r_t < r$, so that there is a bottom region, then

$$x_b = \left\lfloor \frac{w}{c+1} \right\rfloor \tag{18}$$

$$c_{b0} = (x_b + 1)(c + 1) - w \tag{19}$$

$$y_b = \left\lfloor \frac{h - h_t}{r - r_t} \right\rfloor \tag{20}$$

$$r_{b0} = (y_b + 1)(r - r_t) - (h - h_t) \tag{21}$$

Note that the second argument of the "max" in Assignment (13) can be calculated in C as (`h*c*rt + s/2)/s`, thereby avoiding the need for floating-point operations.

E. Analysis

Here we show that the partitioning algorithm described in Section V.D always produces a valid partition (especially, that no segments have a width or height of zero) as long as the area of the rectangle being partitioned is at least as large as the desired number of segments. We also provide some support for our claim that segments tend to be nearly square and have nearly equal size.

It is easily seen that the value of r produced by the algorithm always will satisfy $1 \leq r \leq s$. It is also true that $r \leq h$. To see the latter, first note that if $h > (s-1)w$ so that $r = s$, then $h > (s-1)w \geq s-1$ so that $h \geq s = r$. Otherwise, Condition (10) is satisfied, and multiplying Condition (9) by the left-hand inequality of Condition (10) yields $(r-1)rws < h^2ws$. Dividing by ws results in $(r-1)r < h^2$. Since r and h are integers, this again implies $r \leq h$.

Heuristically, the method of picking r is motivated by the desire to produce nearly square segments. If there are r rows, then the average height of a row is h/r. The average number of columns is s/r, so the average width of a column is wr/s. Setting the average width equal to the average height gives $wr^2 = hs$; hence Condition (10).

Assignments (11) and (12) amount to partitioning s into r divisions, r_t of which have size c and the remainder have size $c + 1$, where $1 \leq r_t \leq r$. Since $r \leq s$, none of the divisions has size zero.

The expression for h_t, Assignment (13), is motivated by the idea that the segments in the top region should have about the same average area as the segments in the bottom region. However, the assignment $h_t = \lfloor hcr_t/s + 1/2 \rfloor$ would not ensure that $h_t \geq r_t$ (for example, if $w = 2$, $h = 7$, and $s = 9$); thus, we compute h_t as the maximum of r_t and $\lfloor hcr_t/s + 1/2 \rfloor$.

Assignments (14) and (15) partition w into c divisions, c_{t0} of which have width x_t and the remainder have width $x_t + 1$. If $r_t < r$ (that is, the bottom region is present), then Assignments (18) and (19) partition w into $c + 1$ divisions, c_{b0} of which are of size x_b and the remainder of which are of size $x_b + 1$. To confirm that all of these divisions have positive width, we need to verify that $c \leq w$ and that, if $r_t < r$, then $c + 1 \leq w$. This is established as follows:[11]

If $h > (s - 1)w$ so that $r = s$, then $c = 1$ and $r_t = r$ by Assignments (11) and (12), so we immediately have the desired result. Otherwise, Condition (10) holds. The right-hand inequality of Condition (10) can be written as

$$\frac{s}{r} \leq \frac{r+1}{h}w$$

and Condition (9) can be written as

$$\frac{s}{r} \leq \frac{h}{r}w$$

Since either $r + 1 \leq h$ or $r \geq h$, it follows that $s/r \leq w$. Thus, $c = \lfloor s/r \rfloor \leq w$ as desired. If $r_t < r$, then s is not a multiple of r, so $s/r \leq w$ implies $s/r < w$, and we have $c + 1 = \lfloor s/r \rfloor + 1 \leq w$.

Assignments (16) and (17) partition the top region (which has height h_t) into r_t rows, r_{t0} of which have height y_t and the remainder have height $y_t + 1$. As previously noted, Assignment (13) ensures $h_t \geq r_t$, so that $y_t \geq 1$.

If $r_t < r$, then Assignments (20) and (21) partition the bottom region (of height $h - h_t$) into $r - r_t$ rows, r_{b0} of which have height y_b and the remainder have height $y_b + 1$. To confirm that $y_b \geq 1$, we must verify that $h - h_t \geq r - r_t$ when $r_t < r$.

There are two cases to consider. First, suppose $h_t = r_t$. We have already noted that $r \leq h$; thus, it follows that $h - h_t \geq r - r_t$, as desired. In the second case, $h_t > r_t$, so Assignment (13) results in $h_t = \lfloor hcr_t/s + 1/2 \rfloor$, from which we have $h_t \leq hcr_t/s + 1/2$ or, equivalently,

$$hr_t \geq \left(h_t - \frac{1}{2} \right) \frac{s}{c} \tag{22}$$

Observe that $s = cr + r - r_t$ from Assignment (12) and, therefore, $s/c > r$. Thus, Inequality (22) implies

$$hr_t > \left(h_t - \frac{1}{2} \right) r$$

Subtracting $(h_t - 1/2)r_t$ from each side yields

---

[11] This elegant line of reasoning is essentially due to Ken Andrews and Sam Dolinar, Jet Propulsion Laboratory, Pasadena, California.

$$\left(h - h_t + \frac{1}{2}\right) r_t > \left(h_t - \frac{1}{2}\right) (r - r_t)$$

so

$$\frac{h - h_t + 1/2}{r - r_t} > \frac{h_t - 1/2}{r_t}$$

Since $h_t > r_t$ and both h_t and r_t are integers, this implies

$$\frac{h - h_t + 1/2}{r - r_t} > 1$$

and we see that

$$\frac{h - h_t}{r - r_t} \geq 1$$

as desired.

VI. Controlling Image Quality and Amount of Compression

In lossy image compression, there is an inherent trade-off between the amount of compression and the quality of the reconstructed image. This trade-off depends on the image content, which can vary widely, and also on the compression parameter values (e.g., wavelet filter choice, number of error-containment segments, etc., in the case of ICER). Because of this variation, providing a satisfactory method of meeting constraints on compressed data volume and/or image quality can be tricky.

Compressors that are not progressive generally provide a single parameter to control image quality, with no direct means of meeting a constraint on compressed data volume. For example, this is typical of JPEG compressors.[12] By contrast, a progressive compression algorithm easily can meet a rate constraint because compression simply can stop once a rate target is met, or the compressed bitstream can be truncated to the desired length. All of the available transmission or storage capacity thus can be used to maximize image quality.

ICER provides two parameters that together are the primary means of controlling image quality and amount of compression: a *byte quota*, indicating a rough maximum number of compressed bytes to produce, and a quality goal parameter that tells ICER to stop when a simple image-quality criterion is met. ICER stops producing compressed bytes once the quality goal or byte quota is met, whichever comes first.

When choosing the values of these parameters, if the primary concern is the compressed data volume, one can set the quality goal to lossless and the compressed image size essentially will be determined by the byte quota. At the other extreme, where the primary concern is the image quality, one can specify a sufficiently large byte quota, and the compressed image quality will be determined by the quality goal. In general, of course, one might select both parameters without knowing beforehand which will be the limiting factor.

---

[12] The JPEG compressor used on the Mars Pathfinder mission did optionally allow a compression ratio to be specified; however, this capability was implemented using a potentially time-consuming iterative procedure that typically compressed an image multiple times to determine a corresponding quality setting that approximately achieved the requested amount of compression.

A. The Quality Goal Parameter

The parameter that determines the quality goal is called the *minimum loss* parameter. This term is used because a smaller value of this parameter indicates a higher quality goal. The minimum loss parameter is a nonnegative integer that determines a minimum number of bit planes that will not be encoded in each subband. Of course, ICER also will stop compressing if it reaches the byte quota. Conversely, if the quality goal is reached, the actual number of compressed bytes produced may be less than the byte quota.

A minimum loss value of M means that compression will stop before the last M bit planes of the level-1 HH subband are encoded. Similarly, bit planes with sufficiently low importance in other subbands are not encoded, taking into account the relative importance of the different subbands (described in Section III.A). The offsets used to determine the relative importance of the bit planes in the subbands are shown in Fig. 18, which is derived from Fig. 7.

For example, if the byte quota is sufficient, then a minimum loss value $M = 0$ will yield lossless compression; $M = 1$ will result in the inclusion of all subband bit planes except the least-significant bit plane of the level-1 HH subband in the compressed output; $M = 2$ will cause the inclusion of all but 2 bit planes of the level-1 HH subband and all but 1 bit plane of each of the level-1 LH, the level-1 HL, and the level-2 HH subbands; etc. When the original image has a dynamic range of B bits per pixel, D stages of wavelet decomposition are used, and $M \geq B + D$, then little or no bit-plane information will be encoded.[13] Note that if all but k bit planes of a subband are encoded, the pixels of this subband are in effect quantized with a step size of 2^k (see Section III.A). Figure 19 shows details from reconstructed images corresponding to several different values of the minimum loss parameter.

Because ICER uses an invertible integer wavelet transform, lossless compression is achieved if the minimum loss parameter is set to 0 and the byte quota is large enough to accommodate all of the compressed subband bit planes. Of course, when ICER is being used to compress images acquired remotely, one often must select the byte quota with little knowledge of image content. Given the wide variation in the compressibility of images, it can be difficult to estimate the minimum byte quota needed to accommodate lossless compression. Often a sensible strategy for achieving lossless compression is to choose a byte quota equal to the number of bytes that would be required to store the original image.

Fig. 18. Relative importance of subbands for three stages of wavelet decomposition. If the minimum loss value minus the number shown is positive, then this difference equals the number of bit planes of the subband that will not be encoded no matter how large the byte quota is.

---

[13] The mean value of a segment's LL subband is encoded in the segment's header even when no bit planes are encoded.
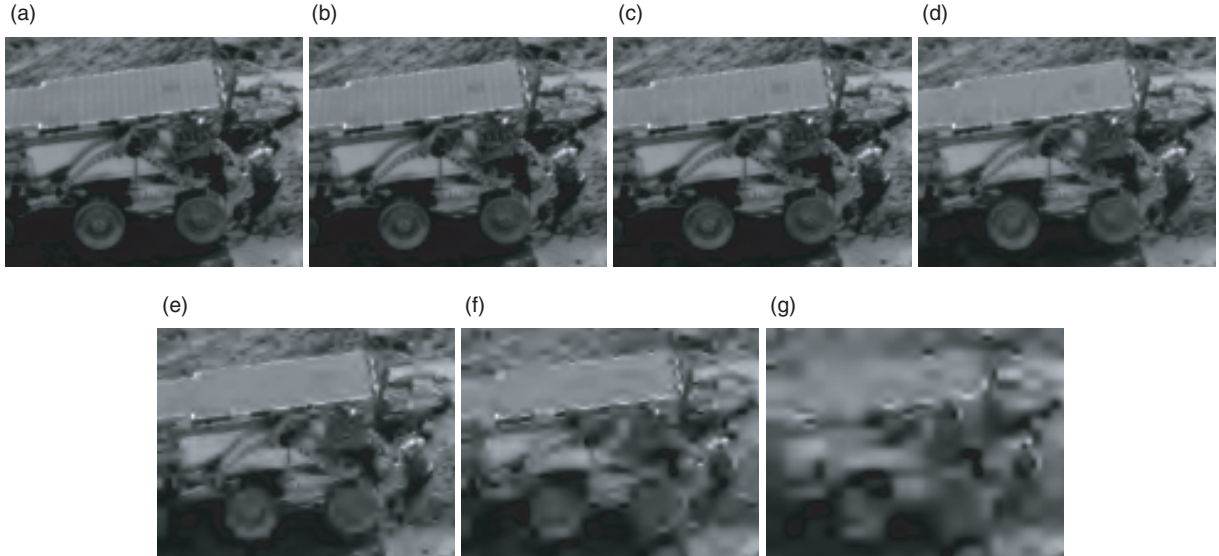
Fig. 19. A 121 × 93 detail from a sample image compressed to different values *M* of the minimum loss parameter. The original image is a 12-bit/pixel Mars Pathfinder image with dimensions 256 × 248: (a) *M* = 0 (8.22 bits/pixel), (b) *M* = 6 (2.81 bits/pixel), (c) *M* = 7 (1.86 bits/pixel), (d) *M* = 8 (1.06 bits/pixel), (e) *M* = 9 (0.51 bits/pixel), (f) *M* = 10 (0.21 bits/pixel), and (g) *M* = 11 (0.09 bits/pixel). In each case, the image is compressed using filter B, 3 stages of decomposition, and 3 error-containment segments.

Figure 20 shows MSE distortion as a function of the minimum loss value for the six Mars surface images described in Section VII; here we use wavelet filter B with four stages of decomposition. Note that the minimum loss parameter provides somewhat coarse control over image quality. We see from the figure that the minimum loss value, when sufficiently small, determines the approximate MSE distortion obtained reasonably well. Figure 21 shows MSE, averaged over all six test images, as a function of the minimum loss value when different wavelet filters are used. Clearly the relationship between minimum loss value and MSE distortion is affected by the choice of wavelet filter. However, it would be a mistake to infer a hierarchy of filters from Fig. 21, since the results give no indication of the compressed data volume required to produce any of the reconstructions, and qualitative differences in the reconstructed images can be more significant than the small differences in MSE anyway. The number of stages of wavelet decomposition also affects the MSE distortion corresponding to minimum loss parameter values, but the effect is tiny.

B. The Byte Quota Parameter and Compressed Data Volume

The byte quota parameter indicates roughly the maximum number of compressed bytes to be produced by ICER. For convenience and compression speed, ICER checks whether the byte quota has been reached only between compression of segments of subband bit planes. Furthermore, the backlog of words remaining in the circular buffers of the segments' interleaved entropy coder instances are not flushed (see Section IV.C) until after the byte quota or the quality goal is reached, and ICER doesn't account for this encoding cost when performing the byte quota check. Consequently, when the quality goal is not reached, the number of output bytes almost always exceeds the byte quota.

The size of this overshoot depends on the number of segments due to two competing effects: a larger number of segments tends to create a larger total backlog of compressed bytes in the interleaved entropy coders for the segments, but having a larger number of segments also causes ICER to check more frequently to see if the byte quota has been reached (as does a larger number of stages of wavelet decomposition). The expected overshoot tends to be minimized when a moderate number of error-containment segments
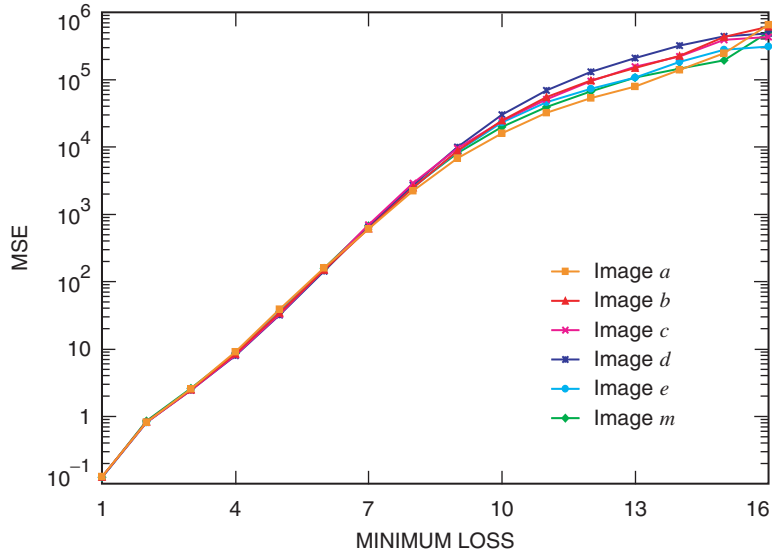
Fig. 20. MSE as a function of minimum loss for the Mars surface test images described in Section VII, using wavelet filter B and four stages of decomposition.
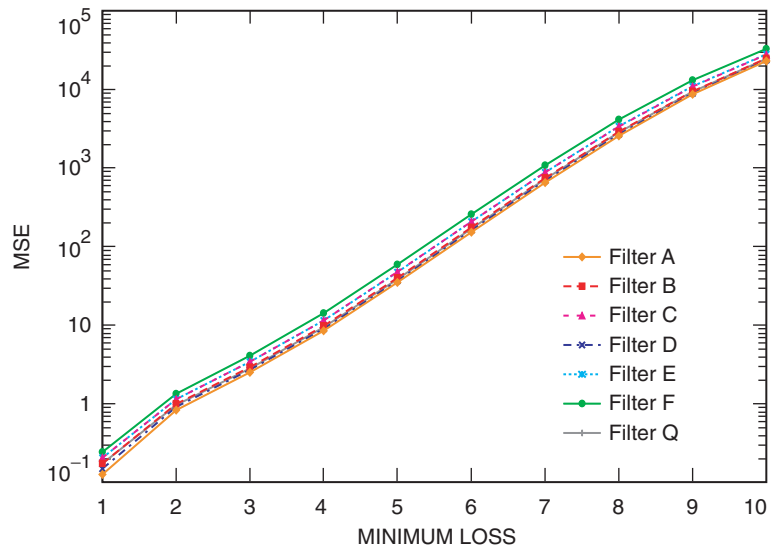

Fig. 21. MSE averaged over the six Mars surface test images described in Section VII, as a function of minimum loss for the different wavelet filters. Four stages of wavelet decomposition are used in each case.

is used and to increase if the size of the circular buffers used in entropy coding is increased. Figure 22 shows an example of the total overshoot and the contribution to overshoot due to the interleaved entropy coder backlog, as a function of the byte quota. Here eight error-containment segments are used. In this example, the overshoot generally is relatively small and does not have a simple dependence on the byte quota.

If desired, one can truncate the compressed output to the desired number of bytes, as described below. We note that, for a given image, compression is faster when fewer compressed bytes are produced (see Section VII.C for a concrete example), and, thus, when the output is truncated to meet the byte quota exactly, the byte overshoot effectively amounts to a slight compression time penalty.
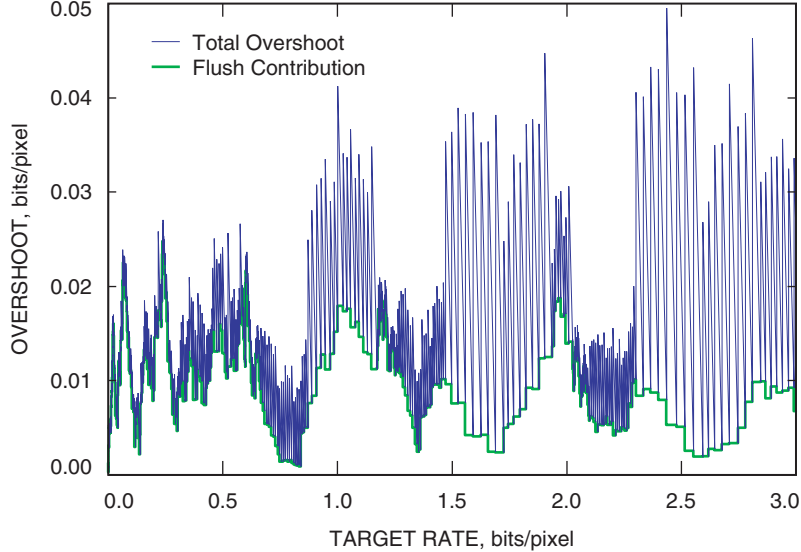
36

Fig. 22. Overshoot for 1024 × 1024 Mars surface test image _m_, described in Section VII, using 8 segments, filter B, and 6 stages of wavelet decomposition. The target rate is the byte quota value converted to bits/pixel. The lower curve shows the contribution to overshoot due to flushing the interleaved entropy coder instances.

Unfortunately, truncating the compressed output can result in a disparity in the reconstructed qualities of the various segments. This effect is due to the variation in the backlog of bits among the segments' interleaved entropy coders. If the average compressed data volume per segment is very small, the quality disparity between the segments can be quite noticeable.

During compression, ICER accumulates output sequentially, but because ICER compresses each error-containment segment of a subband bit plane before moving on to another subband bit plane, the output bitstream interleaves data from the segments. Thus, a compressed segment consists of several blocks (generally one for each subband bit-plane segment plus a header block) that are interleaved in ICER's output. This organization of the output bitstream represents progressive compression across all of the segments, and so is a convenient form from which to extract a specific number of compressed bytes. Thus, if truncation is needed to meet a storage or downlink constraint, one should truncate the overall compressed data stream.

Figure 23(a) illustrates the arrangement of blocks of ICER compressed output that might arise when ICER compresses an image using three segments. In this example, ICER has produced a total of nine blocks; in a typical application, we might more realistically expect hundreds of blocks. Here segment 0 consists of blocks a, d, and g; segment 1 consists of blocks b, e, and h; and segment 2 consists of blocks c, f, and i.

Whether or not truncation is performed, the compressed bitstream should be rearranged prior to transmission so that portions corresponding to a given segment are concatenated in order. This organization helps to minimize the impact of data loss by making it less likely that a single frame loss will affect more than one segment; see Section V.A. Figure 23(b) illustrates the rearrangement of the compressed bitstream in conjunction with truncation to a byte limit. The location of boundaries between blocks is unimportant and need not be recorded once the blocks are arranged as in Fig. 23(b).
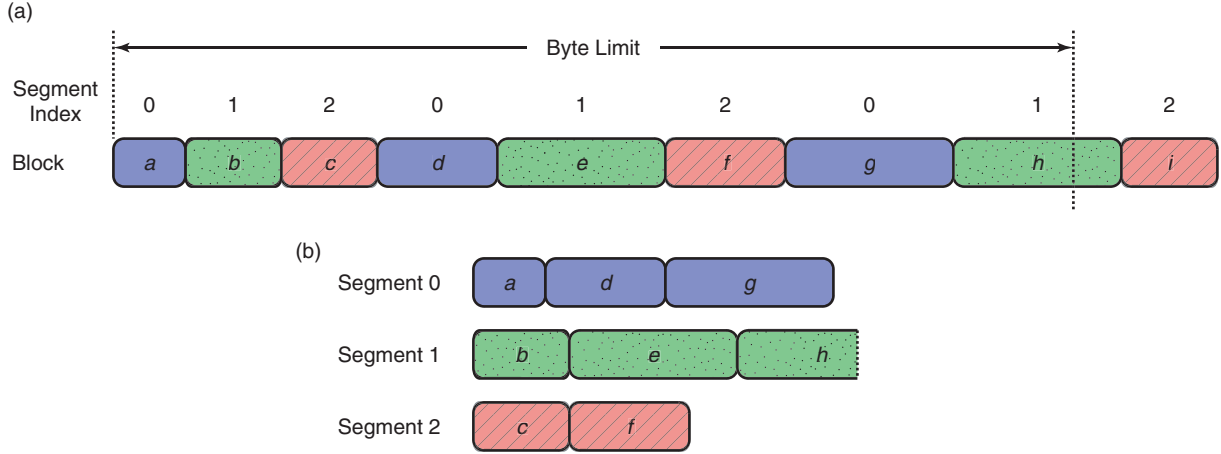
(a)

(b)

Fig. 23. Example of (a) ICER-compressed data output organization and (b) reorganization of compressed blocks following truncation to a byte limit.

VII. Results

In this section, we compare the lossy and lossless compression effectiveness of ICER to that of other image compressors and provide compression speed results for the MER implementation of ICER.

For our comparisons, we use six Mars surface images obtained from the Mars Pathfinder (MPF) mission Imager for Mars Pathfinder (IMP) camera. As shorthand, we use a single letter identifier for each image. Images a through e are single-frame IMP images that were compressed losslessly onboard Mars Pathfinder; these can be obtained from the Planetary Data System (PDS).[14] Image m is a mosaic of several single-frame IMP images, with a larger size (1024 × 1024) that matches the cameras onboard MER. All of the images have a bit depth of 12 bits/pixel. The image dimensions and MPF image identification (ID) numbers are listed in Table 11.

We evaluated the compression performance of ICER, JPEG 2000 [14], JPEG [31], LOCO [3–5], and two different Rice compressors [32]. ICER and JPEG 2000 provide lossless and lossy compression. JPEG provides only lossy compression, and LOCO and Rice provide only lossless compression.

Table 11. MPF images used for evaluations.

Image	MPF image ID	Size ($W \times H$), pixels
a	0053140005	256 × 248
b	0182010121	256 × 248
c	0184010102	256 × 248
d	0182010070	256 × 248
e	0033030101	256 × 248
m	(Mosaic)	1024 × 1024

---

[14] Specifically, we obtained these images from the PDS Planetary Image Atlas for Mars Pathfinder IMP Images, http://www-pdsimage.jpl.nasa.gov/cgi-bin/MPF/MPF_search.pl.

For ICER compression results, we used wavelet filter B, with 4 stages of decomposition on images a through e and 6 stages of decomposition on the larger image m. We used a single error-containment segment in each case.

We obtained JPEG 2000 performance results using the JasPer software[15] with the JPEG-2000 code stream (JPC) compressed image format. As with ICER, we used JasPer with 4 stages of decomposition on images a through e and 6 stages of decomposition on image m. In accordance with the JPEG-2000 standard, the JasPer software provides both an "integer" and a "real" mode that differ primarily in the wavelet transform stage. In the integer mode, JasPer uses a 5/3 reversible integer wavelet transform [9,33]. In the real mode, a floating-point 9/7 wavelet transform [34] is used; it often gives slightly better compression effectiveness (in terms of quantitative measures of image quality), but it requires the use of floating-point operations and a quantization step, and it cannot be used for lossless compression.

JPEG compression performance results were obtained using the JPEG software from the Independent JPEG Group (IJG), compiled to operate on 12-bit source images.[16]

The LOCO software that we tested was implemented by the authors and will be used by MER for lossless compression of 12-bit and 8-bit images. This software uses a variant of the algorithm described in [5], which is itself a modified version of the LOCO-I algorithm of [3,4]. The software that we tested allows the image to be segmented for error-containment purposes, but, as in our tests on ICER, we used a single segment.

The first of the two Rice compressors for which we have results is the variation adopted as a CCSDS standard [35].[17] In our tests we used a block size of 16 samples, a reference interval of 128 blocks, and a segment length of 64 blocks. For the single-frame IMP test images (a through e), we also give compression performance of the Rice compressor used by MPF for lossless image compression. These results are taken directly from the PDS image archive data.

A. Lossy Compression Performance

Rate-distortion performance results for the test images are given in Fig. 24. For our objective distortion measure, we use peak signal-to-noise ratio (PSNR), defined as

$$20 \log_{10} \left(\frac{2^b - 1}{\sqrt{\text{MSE}}} \right) \quad \text{(dB)}$$

where b is the number of bits/pixel in the original image ($b = 12$ for all of the test images here), and MSE is the mean-squared error between the original and reconstructed image. The figure shows that ICER provides rate-distortion performance that is competitive with that of JPEG 2000 and noticeably better than that obtained by JPEG. Since the MPF mission relied on a version of JPEG for lossy image compression, the improvement that ICER offers over JPEG is worth noting as it gives some indication of the benefit that ICER provides to the MER mission.

In Fig. 24, the points shown on the curves for ICER are produced by compression controlled by the minimum loss parameter (i.e., all subband bit planes having some fixed priority value have just been compressed). A plot of MSE distortion versus rate would show the analogous points connected by

---

[15] JasPer software Version 1.600.0.0, `http://www.ece.uvic.ca/~mdadams/jasper/`.

[16] IJG software Version 6b, `ftp://ftp.uu.net/graphics/jpeg/`. Note that the IJG software reverses the byte order of each 2-byte pixel in Portable Gray Map (PGM) format image files.

[17] Our Rice (CCSDS) compression results are obtained from Version 1.0 of the CID_rice software, which is a software implementation of the CCSDS standard, and is produced by the European Space Agency's European Space Research and Technology Center (ESTEC), `http://www.estec.esa.nl/tech/datacwg/lossless.htmls`.
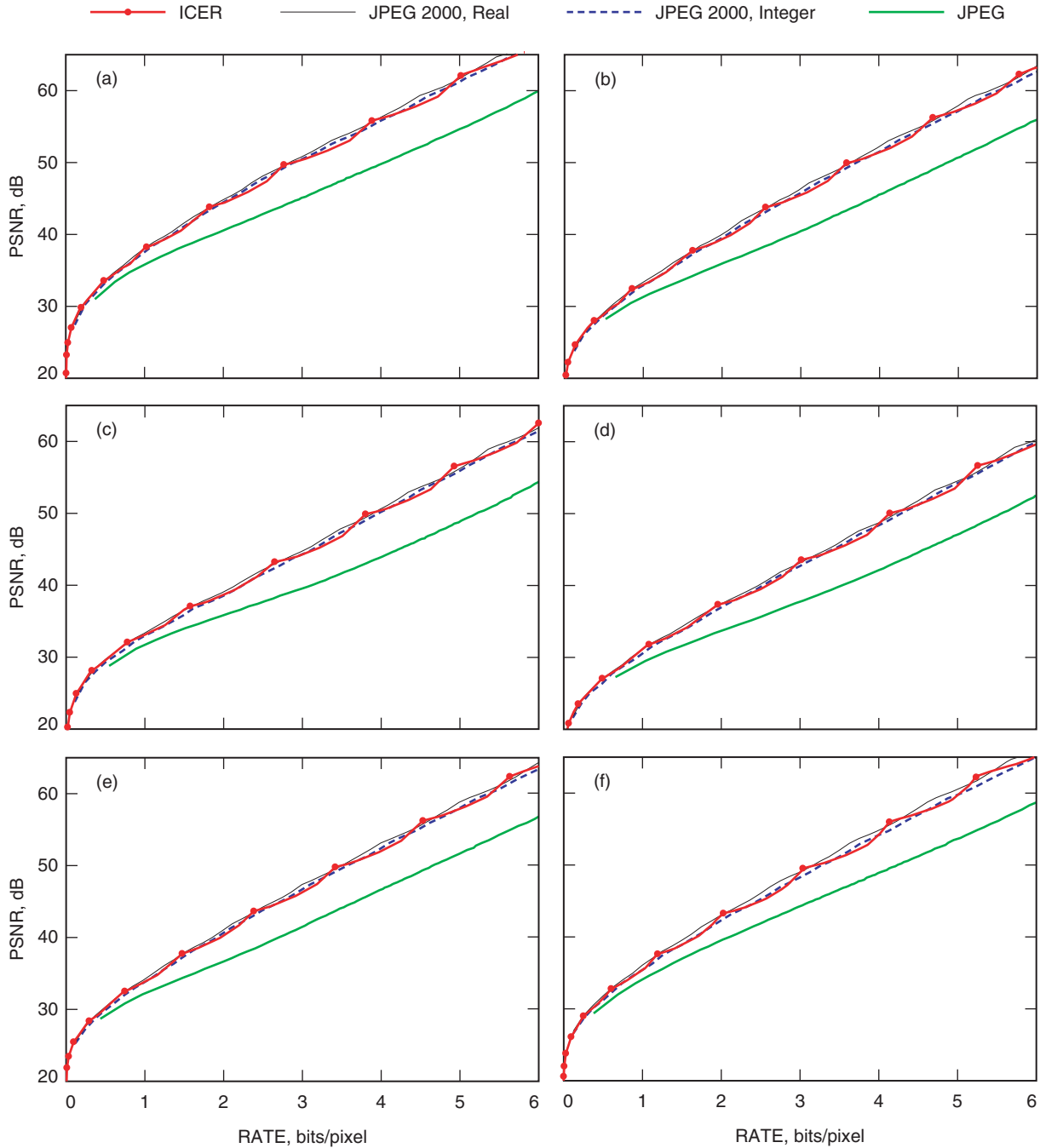
Fig. 24. Rate-distortion performance of ICER, real and integer modes of JPEG 2000, and JPEG on the test images: (a) image *a*, (b) image *b*, (c) image *c*, (d) image *d*, (e) image *e*, and (f) image *m*.

approximately straight line segments, reflecting the sequential nature of the compression within bit planes of a given priority. Thus, when PSNR is plotted, the observed "scalloping" effect occurs. This effect is not noticeable with JPEG 2000; two reasons for this are (1) in JPEG 2000 the multiple coding passes within a bit plane are designed so that bits coded in earlier passes give a larger reduction in distortion per compressed bit [13] and (2) the wavelet transforms used to obtain the JPEG 2000 results produce few bit planes with equal priorities.

B. Lossless Compression Performance

Table 12 gives the lossless compression performance on the 12-bit test images. In addition to losslessly compressing 12-bit images, MER will also losslessly compress 8-bit images formed by quantizing the pixels of 12-bit images. To illustrate lossless compression performance on 8-bit images, we produced an 8-bit version of each test image by linearly scaling each pixel according to the minimum and maximum pixel values in the image to make full use of the 8-bit dynamic range. Table 13 lists the results.

We see that in terms of lossless compression effectiveness, ICER is comparable to LOCO; this is significant because LOCO is a state-of-the-art compressor designed exclusively for lossless image compression. We also observe that ICER's lossless compression effectiveness is superior to that of JPEG 2000 and the Rice compressors, especially the Rice implementation used by the MPF mission.

C. Compression Speed

To provide a rough indication of compression speed, timing results were obtained for the MER implementation of ICER under the VxWorks operating system running on a 20-MHz RAD6000 processor identical to those used by the MER rovers. Figure 25 contains a representative sample of the results. The figure demonstrates the fact that, for a given image, compression is faster when fewer compressed bytes

Table 12. Lossless compression performance on 12-bit images.

Image	Rate, bits/pixel				
	ICER	LOCO	JPEG 2000	Rice (CCSDS)	Rice (MPF)
a	8.19	8.15	8.48	8.63	9.38
b	8.90	8.92	9.24	9.39	10.13
c	9.08	9.10	9.43	9.66	10.21
d	9.40	9.41	9.74	9.93	10.61
e	8.76	8.83	9.11	9.23	9.87
m	8.36	8.62	8.59	9.34	n/a
Average	8.78	8.84	9.10	9.36	—

Table 13. Lossless compression performance on 8-bit images.

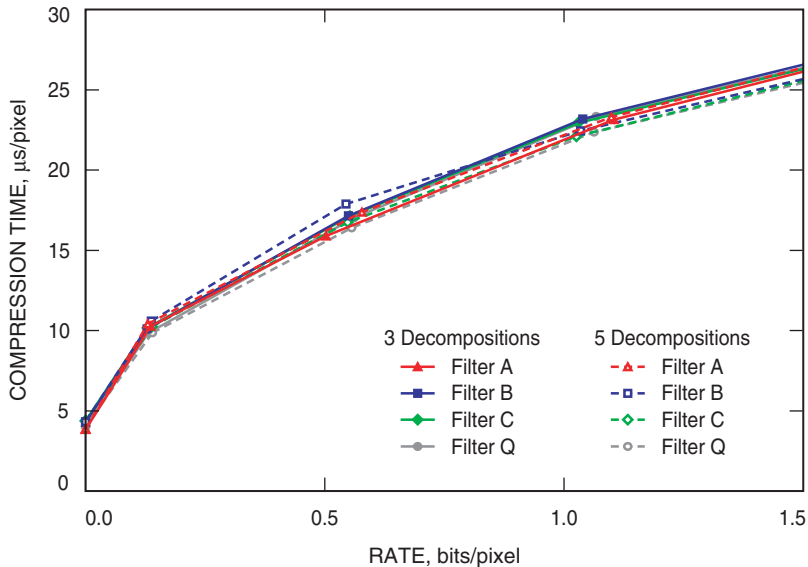| Image	Rate, bits/pixel			
	ICER	LOCO	JPEG 2000	Rice (CCSDS)
a	4.72	4.63	4.86	5.07
b	4.99	4.94	5.17	5.40
c	5.45	5.41	5.67	5.95
d	5.76	5.71	5.98	6.21
e	4.83	4.79	5.00	5.23
m	4.40	4.62	4.64	5.26
Average	5.02	5.02	5.22	5.52

Fig. 25. ICER compression time on the 20-MHz RAD6000 processor used by the MER rovers, for test image *m* using one error-containment segment. Results courtesy of Todd Litwin, Jet Propulsion Laboratory, Pasadena, California.

are produced. The time to produce a compressed image at a bit rate of zero primarily reflects the time required to perform the wavelet decomposition of the image. Observe that the choice of wavelet filter or number of decompositions has little impact on compression speed.

VIII. Conclusion

The ICER image compressor was designed to meet the specialized needs of deep-space applications. ICER is wavelet-based and produces progressive compression, providing lossless and lossy compression, and incorporates an error-containment scheme to limit the effects of data loss on the deep-space channel. ICER achieves state-of-the-art compression effectiveness, providing lossy compression performance competitive with the JPEG 2000 image compression standard, and lossless compression performance competitive with the LOCO image compressor. ICER noticeably outperforms the JPEG image compressor used by the MPF mission and provides significantly more effective lossless compression than the Rice compressor used by that mission.

The MER mission, with a total of 18 cameras on two rovers, will rely heavily on ICER to enable delivery of image data back to Earth during the 180 Martian days of surface operations. The MER mission is significantly advancing the state of practice of image compression for deep-space missions by using image compressors that provide substantially more effective compression than that obtained by the MPF mission.

Rapid advances in imaging technologies will enable future missions to collect even higher volumes of image data and continue to push the need for innovative data compression technologies. We note some possible improvements to ICER that would increase its functionality and effectiveness for future missions:

> (1) Improvements to the overall compression effectiveness of ICER should be possible. A likely avenue is to improve the context modeling and prediction during encoding of bit layers by developing a more refined context modeler. Improvements in compression effectiveness would need to be balanced with complexity considerations.

(2) It would be fairly straightforward to modify ICER to provide more fine-grained control over image-quality requests than currently supported by the minimum loss parameter described in Section VI.A.

(3) The quality disparity that can occur when the bitstream is truncated and the overshoot of the byte quota (both discussed in Section VI.B) are two issues that are largely related to the backlog of bits in each segment's interleaved entropy coder. One straightforward way of reducing the byte overshoot would be to estimate the backlog of bits in each segment's interleaved entropy coder when checking the byte quota. One also could periodically flush each interleaved entropy coder, but this would reduce compression effectiveness slightly. A more integrated approach would involve changing the way that entropy coding is performed.

(4) The decompressor could be modified to incorporate more sophisticated techniques for selecting quantizer reconstruction points than the current method of selecting a point close to the midpoint of each quantizer bin (see Section III.A). A thorough study of the distribution of subband data in typical images could yield an improved approach, perhaps by adaptively adjusting the quantizer reconstruction point based on observations of nearby pixels. This would reduce distortion in the reconstructed image at a given bit rate without changing the compressor. Since the decompressor operates on the ground, fairly complex approaches could be considered.

Progressive compressors like ICER will enable future missions to employ sophisticated data-return strategies involving incremental image-quality improvements to maximize the science value of returned data using an onboard buffer. Improvements in compression techniques and onboard hardware will allow future missions to reap larger benefits from image compression.

Acknowledgments

References

[1] S. Dolinar, A. Kiely, M. Klimesh, S. Shambayati, A. Ortega, S. Lee, P. Sagetong, H. Xie, and R. Manduchi, "Region-of-Interest Data Compression with Prioritized Buffer Management (II)," *Proceedings of the 2002 Earth Science Technology Conference*, paper PS2P2, Pasadena, California, June 11–13, 2002.

[2] J. N. Maki, J. F. Bell, K. E. Herkenhoff, S. W. Squyres, A. Kiely, M. Klimesh, M. Schwochert, T. Litwin, R. Willson, A. Johnson, M. Maimone, E. Baumgartner, A. Collins, M. Wadsworth, S. T. Elliot, A. Dingizian, D. Brown, E. C. Hagerott, L. Scherr, R. Deen, D. Alexander, and J. Lorre, "The Mars Exploration Rover Engineering Cameras," *Journal of Geophysical Research—Planets*, vol. 108, no. E12, 8071, 2003 (in press).

[3] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, August 2000.

[4] M. J. Weinberger, G. Seroussi and G. Sapiro, "LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm," *Proceedings of the IEEE Data Compression Conference*, pp. 140–149, 1996.

[5] M. Klimesh, V. Stanton, and D. Watola, "Hardware Implementation of a Lossless Image Compression Algorithm Using a Field Programmable Gate Array," *The Telecommunications and Mission Operations Progress Report 42-144, October–December 2000,* Jet Propulsion Laboratory, Pasadena, California, pp. 1–11, February 15, 2001.
http://tmo.jpl.nasa.gov/tmo/progress_report/42-144/144H.pdf

[6] A. Said and W. Pearlman, "Reversible Image Compression via Multiresolution Representation and Predictive Coding," *Proceedings of the SPIE Conference on Visual Communications and Image Processing 1993*, Cambridge, Massachusetts, *Proc. SPIE*, vol. 2094, pp. 664–674, November 1993.

[7] A. Said and W. A. Pearlman, "An Image Multiresolution Representation for Lossless and Lossy Compression," *IEEE Transactions on Image Processing*, vol. 9, no. 5, pp. 1303–1310, September, 1996.

[8] A. Zandi, A. D. Allen, E. L. Schwartz, and M. Boliek, "CREW: Compression with Reversible Embedded Wavelets," *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, pp. 212–221, March 1995.

[9] M. D. Adams and F. Kossentini, "Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1010–1024, June 2000.

[10] S. G. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, July 1989.

[11] J. M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, December 1993.

[12] A. Said and W. Pearlman, "A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1993.

[13] D. Taubman, "High Performance Scalable Image Compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, July 2000.

[14] M. D. Adams, *The JPEG-2000 Still Image Compression Standard*, ISO/IEC JTC 1/SC 29/WG 1 N 2412, September 2001.

[15] E. S. Hong, R. E. Ladner, and E. A. Risken, "Group Testing for Wavelet Packet Image Compression," *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, pp. 73–82, March 2001.

[16] A. B. Kiely, "Progressive Transmission and Compression of Images," *The Telecommunications and Data Acquisition Progress Report 42-124, October–December 1995*, Jet Propulsion Laboratory, Pasadena, California, pp. 88–103, February 15, 1996.
http://tmo.jpl.nasa.gov/tmo/progress_report/42-124/124E.pdf

[17] ISO/IEC, ISO/IEC 14492-1, *Lossy/Lossless Coding of Bi-level Images*, 2000.

[18] J. Rissanen and G. G. Langdon, "Arithmetic Coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, March 1979.

[19] L. Bottou, P. G. Howard, and Y. Bengio, "The Z-Coder Adaptive Binary Coder," *Proceedings of the IEEE Data Compression Conference*, pp. 13–22, March 1998.

[20] P. G. Howard, "Interleaving Entropy Codes," *Proceedings Compression and Complexity of Sequences 1997*, Salerno, Italy, pp. 45–55, 1998.

[21] A. B. Kiely and M. Klimesh, "A New Entropy Coding Technique for Data Compression," *The InterPlanetary Network Progress Report 42-146, April–June 2001*, Jet Propulsion Laboratory, Pasadena, California, pp. 1–48, August 15, 2001.
http://ipnpr.jpl.nasa.gov/tmo/progress_report/42-146/146G.pdf

[22] A. B. Kiely and M. Klimesh, "Memory-Efficient Recursive Interleaved Entropy Coding," *The InterPlanetary Network Progress Report 42-146, April–June 2001*, Jet Propulsion Laboratory, Pasadena, California, pp. 1–14, August 15, 2001.
http://ipnpr.jpl.nasa.gov/tmo/progress_report/42-146/146J.pdf

[23] F. Ono, S. Kino, M. Yoshida, and T. Kimura, "Bi-Level Image Coding with MELCODE—Comparison of Block Type Code and Arithmetic Type Code," *Proc. IEEE Global Telecommunications Conference (GLOBECOM '89)*, pp. 0255–0260, November 1989.

[24] K. Nguyen-Phi and H. Weinrichter, "A New Binary Source Coder and its Application to Bi-Level Image Compression," *Proc. IEEE Global Telecommunications Conference (GLOBECOM '96)*, London, England, pp. 1483–1487, November 1996.

[25] S. W. Golomb, "Run-Length Encodings," *IEEE Transactions on Information Theory*, vol. IT-12, no. 3, pp. 399–401, July 1966.

[26] R. G. Gallager and D. C. Van Voorhis, "Optimal Source Codes for Geometrically Distributed Integer Alphabets," *IEEE Transactions on Information Theory*, vol. IT-21, no. 2, pp. 228–230, March 1975.

[27] Consultative Committee for Space Data Systems, *CCSDS 102.0-B-5: Packet Telemetry*, Blue Book, issue 5, November 2000.
http://www.ccsds.org/documents/102x0b5.pdf

[28] Consultative Committee for Space Data Systems, *CCSDS 101.0-B-6: Telemetry Channel Coding*, Blue Book, issue 6, October 2002.
http://www.ccsds.org/documents/101x0b6.pdf

[29] R. McEliece and L. Swanson, "On the Decoder Error Probability for Reed-Solomon Codes," *IEEE Transactions on Information Theory*, vol. 32, no. 5, pp. 701–703, September 1986.

[30] Consultative Committee for Space Data Systems, *CCSDS 100.0-G-1: Telemetry Summary of Concept and Rationale*, Green Book, issue 1, December 1987.
http://www.ccsds.org/documents/100x0g1.pdf

[31] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, 1993.

[32] R. F. Rice, *Some Practical Universal Noiseless Coding Techniques, Part III, Module PSI14,K+*, JPL-91-3, Jet Propulsion Laboratory, Pasadena, California, November 1991.

[33] A. R. Calderbank, I. Daubechies, W. Sweldins, and B.-L. Yeo, "Wavelet Transforms that Map Integers to Integers," *Applied and Computational Harmonic Analysis*, vol. 5, pp. 332–369, July 1998.

[34] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image Coding using Wavelet Transform," *IEEE Transactions on Image Processing*, vol. 1, no. 2, pp. 205–220, April 1992.

[35] Consultative Committee for Space Data Systems, *CCSDS 121.0-B-1: Lossless Data Compression*, Blue Book, issue 1, May 1997.
http://www.ccsds.org/documents/121x0b1.pdf