An Improved Progressive Lossy-to-Lossless Coding Method for Arbitrarily-Sampled Image Data

Michael D. Adams
Dept. of Electrical and Computer Engineering, University of Victoria
Victoria, BC, V8W 3P6, Canada
mdadams@ece.uvic.ca

***Abstract*—A method for the progressive lossy-to-lossless coding of arbitrarily-sampled image data is proposed. Through experimental results, the proposed method is demonstrated to have a rate-distortion performance that is vastly superior to that of the state-of-the-art image-tree (IT) coding scheme. In particular, at intermediate rates (i.e., in progressive decoding scenarios), the proposed method yields image reconstructions with a peak-signal-to-noise ratio that is much higher (sometimes by several dB) than the IT scheme, while simultaneously achieving a slightly lower lossless rate.**

I. Introduction

In recent years, there has been a growing interest in image representations based on nonuniform sampling [1]–[5]. Since most images are nonstationary in nature, the use of uniform sampling (such as lattice-based sampling) for images is almost always suboptimal, as it tends to place too many sample points in regions of slow variation and too few sample points in regions of rapid change. With nonuniform sampling, the sample points can be adapted to the image content, yielding a more efficient representation. Of course, with the use of datasets that are arbitrarily (i.e., nonuniformly) sampled comes the need to compress such datasets for storage and communication. Moreover, in many applications, it is highly desirable to employ a coding scheme that offers progressive lossy-to-lossless functionality. With such functionality, the decoder need not wait until the entire coded bitstream is received in order to start decoding. Instead, it can start decoding as soon as it receives only a very small portion of the coded bitstream, and as more of the coded bitstream is received, a better and better approximation of the coded dataset can be reconstructed, with a lossless reproduction being obtained after the entire coded bitstream has been decoded.

Although numerous schemes have been proposed for the coding of arbitrarily-sampled image data (e.g., [4], [5]), relatively fewer provide progressive lossy-to-lossless functionality. To date, one of the best performing progressive coding schemes is the so called **image-tree (IT)** coding method proposed in [5]. In this paper, we propose a new tree-based coding method for arbitrarily-sampled image data, which was inspired by the IT scheme. As will be shown later through experimental results, our proposed method has vastly superior progressive coding performance relative to the IT scheme, as well as having slightly improved lossless coding performance.

The remainder of this paper is organized as follows. Section II provides some background information. Section III introduces a new tree-based representation for arbitrarily-sampled image datasets, and then Section IV proposes an effective method for coding the information in this representation. In Section V, the coding efficiency of the proposed method is compared to that of the IT scheme, with the proposed method proving to yield vastly superior progressive coding performance at intermediate rates as well as slightly improved performance at lossless rates. Finally, Section VI concludes the paper with a summary of our key results.

Before proceeding further, a brief digression is in order regarding the notation used herein. The set of integers and set of real numbers are denoted as $\mathbb{Z}$ and $\mathbb{R}$, respectively. The cardinality of a set S is denoted $|S|$. Lastly, for $x \in \mathbb{R}$, $\lfloor x \rfloor$ and $\lceil x \rceil$ respectively denote the largest integer not more than x (i.e., the floor function) and the smallest integer not less than x (i.e., the ceiling function).

II. Background

An image is an integer-valued function f defined for integer points (x, y) in the domain $D = \{0, 1, \ldots, W-1\} \times \{0, 1, \ldots, H-1\}$. Without loss of generality, we assume that the range of f can be represented as P-bit unsigned integers. Such an image can be approximated using a dataset that consists of: 1) a set $S = \{(x_i, y_i)\}_{i=0}^{|S|-1}$ of sample points, with $S \subset \mathbb{Z}^2$; and 2) their corresponding sample values $\{z_i\}_{i=0}^{|S|-1}$, where $z_i = f(x_i, y_i)$. This arbitrarily-sampled dataset is the type of dataset that we are concerned with coding herein. As a matter of terminology, we refer to the quantity $|S| / |D|$ as the **sampling density** of the dataset.

In anticipation of things to come, we now introduce a simple transform relevant to our proposed coding method. The two-point **average-difference (AD) transform** maps two integers x_0 and x_1 to two integers y_0 and y_1, as given by

$$y_0 = \left\lfloor \tfrac{1}{2}(x_0 + x_1) \right\rfloor \quad \text{and} \quad y_1 = x_1 - x_0, \tag{1}$$

where y_0 and y_1 correspond to the approximate average of x_0 and x_1 and the difference between x_0 and x_1, respectively. Due to the form of (1), if x_0 and x_1 can each be represented with n bits (i.e., an n-bit integer), y_0 and y_1 can be represented with n and $n+1$ bits, respectively, a fact that we make use of later. The transform computed by (1) is invertible, with its inverse given by

$$x_0 = y_0 - \left\lfloor \tfrac{1}{2} y_1 \right\rfloor \quad \text{and} \quad x_1 = y_1 + x_0. \tag{2}$$

III. Tree-Based Representation

In order to be able to present our proposed coding method, we must first introduce a tree-based representation of

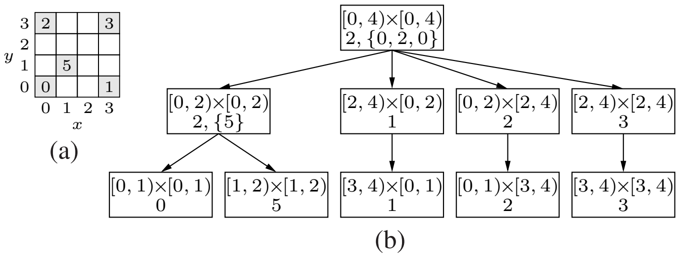


Fig. 1. ADIT example. (a) An arbitrarily-sampled image dataset and (b) its corresponding ADIT representation.

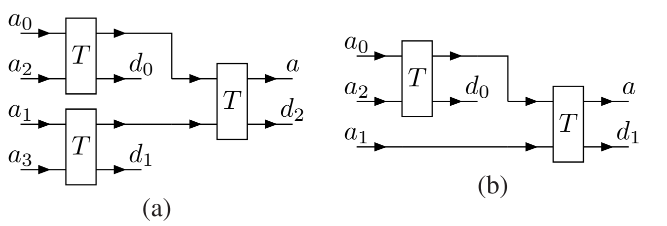


Fig. 2. Computation of the approximation and detail coefficients for a given node in the case that (a) the node has all four possible children; and (b) the node is missing the child corresponding to approximation coefficient a_3.

arbitrarily-sampled datasets, called the **average-difference image tree (ADIT)**. To represent an arbitrarily-sampled dataset, the ADIT must capture: 1) the sample positions (i.e., the position in the image domain of each of the sample points); and 2) the sample values (i.e., the value of the image function at each sample point). The ADIT shares some similarities with the **image tree (IT)** from [5]. The main difference between these two tree-based representations is that they use a completely different approach to capture sample-value information. As it turns out, the way in which the ADIT represents sample-value information facilitates more efficient coding relative to the IT approach.

As suggested above, the ADIT is a tree. Each node in the tree has zero to four children and is associated with the following information: 1) a rectangular region in the image domain, called a **cell**; 2) an approximation coefficient; and 3) $\max\{c-1, 0\}$ detail coefficients, where c is the number of children that the node possesses. The approximation coefficient specifies the approximate average sample value of all samples contained in the node's cell, while the detail coefficients specify the difference between the approximation coefficient of the node and the approximation coefficients of its children. An example of an ADIT is provided in Fig. 1. The ADIT is shown in Fig. 1(b) and the arbitrarily-sampled dataset that it represents is given in Fig. 1(a). In the figure, each node in the ADIT is labelled (in order) with its associated cell, approximation coefficient, and detail coefficients if any (appearing in brace brackets). In what follows, we will explain how the ADIT is constructed for a given arbitrarily-sampled dataset.

First, we consider how to determine the number and connectivity of nodes in the ADIT as well as the relationship between the position of a node in the ADIT and its associated cell. An arbitrary node in the ADIT, with the cell $R = [x_0, x_1) \times [y_0, y_1)$, is associated with four possible children having cells $\{R_i\}_{i=0}^{3}$ given by

$$R_0 = [x_0, x_\mathsf{m}) \times [y_0, y_\mathsf{m}), \quad R_1 = [x_\mathsf{m}, x_1) \times [y_0, y_\mathsf{m}),$$
$$R_2 = [x_0, x_\mathsf{m}) \times [y_\mathsf{m}, y_1), \quad \text{and} \quad R_3 = [x_\mathsf{m}, x_1) \times [y_\mathsf{m}, y_1),$$

where $x_\mathsf{m} = \left\lfloor \frac{1}{2}(x_0 + x_1 + 1) \right\rfloor$ and $y_\mathsf{m} = \left\lfloor \frac{1}{2}(y_0 + y_1 + 1) \right\rfloor$. In other words, the cells $\{R_i\}_{i=0}^{3}$ are obtained by splitting R at its approximate midpoint in each of horizontal and vertical directions. To establish the connectivity of nodes, we start with an ADIT consisting of a single node that is associated with the cell $[0, W) \times [0, H)$. Then, starting from this single root node, the remainder of the nodes are generated by recursively adding, to each node in the ADIT with cell R, any child node whose cell R' satisfies: 1) R' contains at least one sample point; 2) R' has nonzero area; and 3) $R' \neq R$. This process leads to an ADIT with $L = \lceil \log_2 \max\{W, H\} \rceil + 1$ levels, and is associated with a quadtree partitioning of the image domain. Furthermore, the terminal nodes in the ADIT have a one-to-one correspondence with sample points. Each terminal node specifies the location of a single sample point. In particular, the cell of a terminal node always contains exactly one point in $\mathbb{Z}^2$, which must correspond to a sample point. For example, in Fig. 1, one can see that each of the terminal nodes in the ADIT corresponds to a cell containing exactly one integer lattice point which is a sample point.

Now, we examine how the approximation and detail coefficients for a node are calculated. Consider an arbitrary node n with c children, which (as stated above) has one approximation coefficient a and m detail coefficients $\{d_i\}_{i=0}^{m-1}$, where $m = \max\{c-1, 0\}$. For a terminal node (which corresponds to a sample point), a is simply chosen as the sample value of the corresponding sample point, and there are no detail coefficients (since $c = 0$). For a nonterminal node n, the situation is more complicated as a and $\{d_i\}_{i=0}^{m-1}$ are computed through the repeated application of the AD transform given by (1). Let $\{a_i\}_{i=0}^{3}$ denote the approximation coefficients of the four possible children of node n. Suppose that $c = 4$ (i.e., all four possible children are present). In this case, $m = 3$, and a and $\{d_i\}_{i=0}^{2}$ are computed by applying the AD transform repeatedly as shown in Fig. 2(a). In this figure, each block labelled "T" corresponds to the AD transform with the top input, bottom input, top output, and bottom output corresponding to the quantities x_0, x_1, y_0, and y_1 in (1), respectively. Suppose now that $c < 4$. In this case, some of the four possible children are not present and in effect one or more of the $\{a_i\}_{i=0}^{3}$ is missing. To handle missing inputs, we apply the following two rules to the block diagram in Fig. 2(a): 1) any transform block with only one input is replaced by an identity block that does not produce any detail coefficient; 2) if any transform block has no inputs, the block is simply removed altogether. For example, if $c = 3$ and the child corresponding to a_3 is not present, $m = 2$, and a and $\{d_i\}_{i=0}^{1}$ are calculated as shown in Fig. 2(b). Due to the dynamic range properties of the AD transform noted earlier (in Section II), each approximation coefficient can be represented as a P-bit unsigned integer and each detail coefficient can be represented as a $(P+1)$-bit signed integer.

IV. Coding Method

Having introduced the ADIT representation of an arbitrarily-sampled image dataset, we now propose an efficient scheme for coding the information in an ADIT. In what follows, we describe only the encoding process in detail, since the decoding process follows from symmetry. To capture all of the information in the ADIT without redundancy, it is necessary to specify: 1) the width W and height H of the image domain; 2) the **child configuration (CC)** of each node, that is, which of the four possible children are present for each node; 3) the approximation coefficient a_r of the root node; and 4) the **detail coefficients (DCs)**, if any, of each node. Note that the approximation coefficients for the non-root nodes are not required, since these coefficients can be computed from a_r and the detail coefficients.

In addition to the ADIT used to represent the arbitrarily-sampled image dataset, two other key data structures are employed by the encoder: 1) the CC queue and 2) the DC queue. The CC queue is a priority queue that holds nodes whose CC information has not yet been coded. The node priorities are chosen to correspond to a breadth-first traversal of the tree (i.e., so that nodes placed on the CC queue are removed in breadth-first order). The DC queue is a first-in first-out (FIFO) queue that holds nodes whose CC information has been coded but whose DC information has not yet been fully coded. The encoding process proceeds as follows. To begin, a short header containing W, H, P, and a_r is output. Then, the context-adaptive binary arithmetic-coding engine [6], which is used to encode the remainder of the data, is initialized. Initially, the root node is placed on the CC queue. Then, the algorithm alternates between coding CC information for nodes on the CC queue and DC information for nodes on the DC queue, switching queues when the queue currently being processed becomes empty or a maximum entropy budget has been exhausted. After a node on the CC queue is processed, it is moved to the DC queue and each of its children is placed on the CC queue. After a node on the DC queue is processed, it is placed on the DC queue only if it still has more DC information remaining to be coded. The overall encoding process is shown in more detail in Algorithm 1, where the algorithms for encoding CC and DC information are to be specified shortly. As can be seen from Algorithm 1, the entropy budget for the CC queue (i.e., 512 bits) is twice the entropy budget for the DC queue (i.e., 256 bits). This gives a relatively higher priority to sample-position information in the coded bitstream, which was found to be beneficial for good rate-distortion performance (based on significant experimentation). Next, we consider how the CC and DC information is coded. In the case of the CC information, the encoding process is identical to that used in the IT method. So, our focus in what follows is on the coding of DC information.

Binarization. In the coding of DC information, the need arises to encode $(n+1)$-bit signed integers with n bits of magnitude plus a sign bit. In order for these integers to be coded by a binary arithmetic coder, they must first be binarized (i.e., converted to a sequence of binary symbols). To accomplish this, we define a family of binarizations parameterized by n and an integer f, where $f \in [1, n]$, which we denote as $\mathrm{SI}(n, f)$. The $\mathrm{SI}(n, f)$ binarization process behaves identically to the $\mathrm{UI}(n, f)$ binarization process for unsigned integers described in [5], except that, in $\mathrm{SI}(n, f)$ binarization, an extra binary symbol is coded in bypass mode immediately following the coding of the first non-zero magnitude bit of the integer. This extra binary symbol is the sign bit for the integer being coded. (Note that bypass mode is simply an arithmetic-coding context in which both binary symbols have fixed and equal probabilities.)

Algorithm 1 Encoding algorithm.

```
 1: ccBudget := 512
 2: dcBudget := 256
 3: encode header information (i.e., $W$, $H$, $P$, and $a_\mathrm{r}$)
 4: insert root node on CC queue
 5: while CC queue not empty or DC queue not empty do
 6:    while ccBudget $> 0$ and CC queue not empty do
 7:       set node to element at front of CC queue and remove
          element from queue
 8:       encode CC information for node and set b to entropy
          of information just coded
 9:       ccBudget := ccBudget - b
10:       insert each child of node on CC queue
11:       insert node on DC queue
12:    end while
13:    while dcBudget $> 0$ and DC queue not empty do
14:       set node to element at front of DC queue and remove
          element from queue
15:       invoke DC encoding process for node and set b to
          entropy of information just coded
16:       if still more DC data to encode for node then
17:          insert node on DC queue
18:       end if
19:       dcBudget := dcBudget - b
20:    end while
21:    ccBudget := min{512, ccBudget + 512}
22:    dcBudget := min{256, dcBudget + 256}
23: end while
```

DC information coding. Each detail coefficient is a $(P+1)$-bit signed integer, consisting of a P-bit magnitude plus a sign bit, and each detail coefficient for a given node is coded using $\mathrm{SI}(P, \min\{P, 4\})$ binarization conditioned on the node's level in the tree. Each time the DC encoding process is invoked for a particular node, one more magnitude bit is coded (starting from the most-significant bit position) for each of the node's detail coefficients. Whenever the first nonzero magnitude bit is coded for a detail coefficient, it is immediately followed by the sign bit. Since each invocation of the DC coding process codes one magnitude bit from each of the node's detail coefficients and each detail coefficient has P magnitude bits, all of the DC information for a node will be encoded after the coding process has been invoked P times for the node.

Decoding. As mentioned earlier, the decoding process simply mirrors the encoding process. For this reason, we do not describe the decoding process in any depth. There is, however, one detail about decoding that is worth noting. In particular, at intermediate rates during decoding, a terminal node in the partially decoded ADIT may be such that its cell contains more than one integer lattice point. In such a case, the terminal node is deemed to be associated with a sample point located at the centroid of the node's cell.

V. Performance Evaluation

To demonstrate the effectiveness of our proposed method, we compare its coding performance to that of the state-of-the-art IT scheme. For evaluation purposes in our work, 40 images

TABLE I. TEST IMAGES

Name	Size	Bits/Sample	Description
ct	512×512	12	CT scan [8]
lena	512×512	8	woman [7]
peppers	512×512	8	collection of vegetables [7]

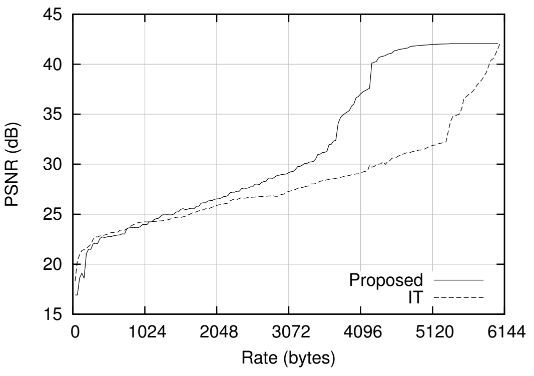
TABLE II. LOSSLESS CODING PERFORMANCE OF THE PROPOSED AND IT METHODS FOR SEVERAL DATASETS

Image	Sampling Density (%)	Rate (Bytes)	
		Proposed	IT
ct	0.5	3266	3279
	1.0	6054	6086
	2.0	11077	11170
	4.0	20106	20300
lena	0.5	2779	2797
	1.0	5098	5115
	2.0	9284	9329
	4.0	16729	16846
peppers	0.5	2827	2833
	1.0	5217	5229
	2.0	9527	9571
	4.0	17180	17285

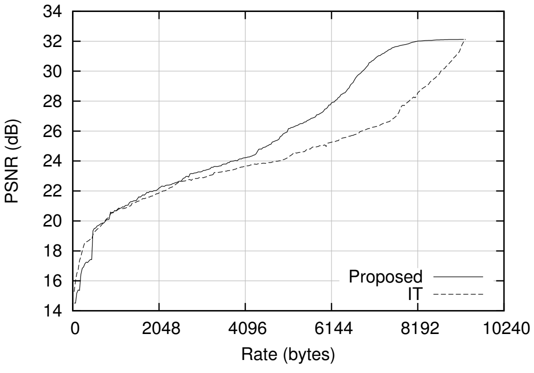
were employed, taken mostly from well known collections including the USC image database [7] and JPEG-2000 test set [8]. Herein, we present results for a small representative subset of these images, namely the three images listed in Table I, which cover both photographic and medical imagery. Since the above images are uniformly-sampled on a rectangular lattice, arbitrarily-sampled datasets had to be generated from these images to be used for evaluation purposes. To produce such datasets, we employed the Delaunay mesh-generation scheme proposed in [2]. From an arbitrarily-sampled dataset, an image reconstruction that is uniformly sampled on a rectangular lattice is synthesized by constructing a piecewise-linear interpolant over the Delaunay triangulation of the dataset's sample points, as described in [2].

For each test image and several sampling densities, we generated an arbitrarily-sampled dataset (using the method of [2] mentioned above). Then, each arbitrarily-sampled dataset was losslessly coded using each of the proposed and IT methods and the lossless rate measured. A representative subset of these lossless coding results is shown in Table II. Next, each of the losslessly coded bitstreams was progressively decoded with the reconstruction error being measured with respect to the original lattice-sampled image as a function of rate. A representative subset of these progressive coding results (for three test cases) is shown in Fig. 3. Each of the three graphs in the figure shows the **peak-signal-to-noise ratio (PSNR)** plotted against the number of bytes decoded. In what follows, we will examine the above results (from Fig. 3 and Table II) more closely.

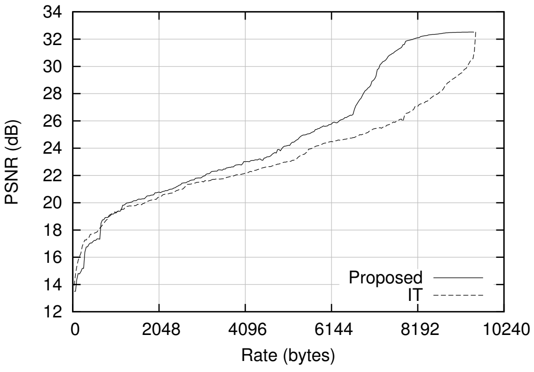
Analysis of progressive coding results. First, let us consider the progressive coding results from Fig. 3. Each plot in this figure shows the PSNR of the reconstructed image plotted against rate. On each graph, the far left corresponds to no information having been decoded, while the far right corresponds to the coded bitstream having been fully decoded and the arbitrarily-sampled dataset having been losslessly reconstructed. As noted above, the PSNR is measured relative to the original lattice-sampled image. Since the arbitrarily-sampled dataset only approximates the original lattice-sampled image, the PSNR does not become infinite (corresponding to zero mean-squared error) when the bitstream is fully decoded. Instead the PSNR reaches a finite maximum value corresponding to the difference between the arbitrarily-sampled dataset and original lattice-sampled image.



(a)



(b)



(c)

Fig. 3. Progressive coding performance of the proposed and IT methods for a dataset corresponding to (a) the ct image at a sampling density of 1%; (b) the lena image at a sampling density of 2%; and (c) the peppers image at a sampling density of 2%.

From the graphs in Figs. 3(a) to (c), it is clear that the proposed method consistently yields image reconstructions with higher PSNR (often by several dB) than the IT method, except at very low rates. The performance at these very low rates, however, is of little interest since the obtained image reconstructions are of such poor quality as to be useless for practical applications. So, in a practical sense (i.e., at rates that yield image reconstructions of sufficient quality to be useful for practical applications), the proposed method consistently outperforms the IT scheme. A much closer examination of the

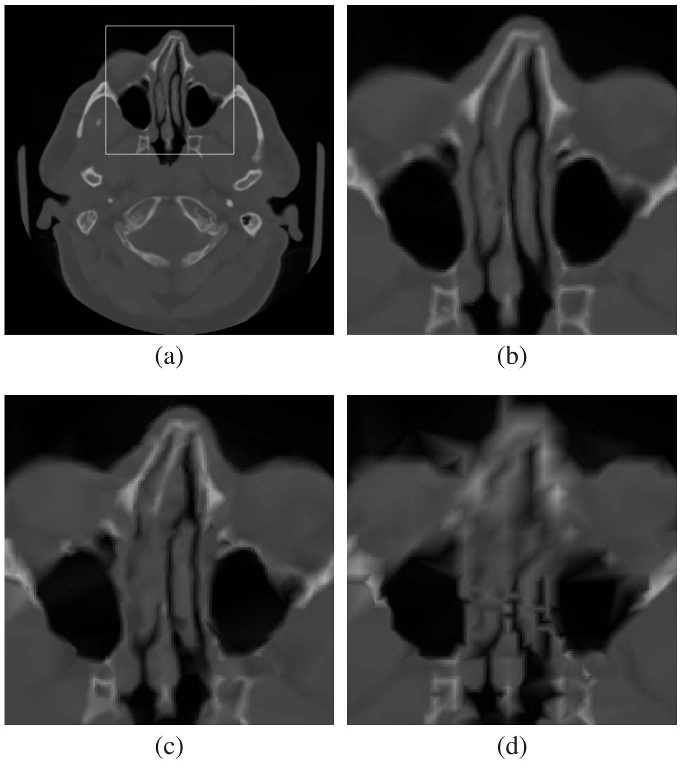


Fig. 4. Lossy coding example for the `ct` image. (a) The original image with a region of interest marked by a rectangle; and (b) the region of interest in the original image. The lossy reconstruction obtained after 3757 bytes (approximately 62%) of the bitstream has been decoded with each of the (c) proposed (33.94 dB) and (d) IT (27.59 dB) methods.

PSNR numbers shows that for the graphs in Figs. 3(a), (b), and (c), the proposed method beats the IT scheme by average and maximum margins of 3.45 dB and 10.88 dB, 1.50 dB and 4.81 dB, and 1.61 dB and 5.64 dB, respectively. These margins are obviously very significant. So, the progressive coding performance of the proposed method is clearly quite superior to that of the IT scheme.

For the most part, PSNR was found to correlate reasonably well with subjective image quality. For the benefit of the reader, however, we provide an example for illustrative purposes. In particular, for the test case considered in Fig. 3(a) (i.e., the dataset for the `ct` image), Fig. 4 shows the reconstructed images obtained at an intermediate rate of 3757 bytes, which corresponds to approximately 62% of coded bitstream having been decoded. Comparing the reconstructed images produced by the proposed and IT methods in Figs. 4(c) and (d), respectively, we can see that reconstruction from the proposed method more faithfully reproduces the original image than the one from the IT scheme.

Analysis of lossless coding results. In each of the graphs in Fig. 3, the rate at which the proposed and IT methods achieve lossless reconstruction is evidently quite close. Now, we will consider the results of Table II in order to compare the lossless coding performance of the two methods more precisely. Examining the results of Table II, we can see that the proposed method performs slightly better than the IT scheme, yielding a lossless rate that is consistently lower by about 0.2% to 1.0%, depending on the particular dataset being coded. Thus, the superior coding performance of the proposed method (over the IT scheme) at intermediate rates does not come at the cost of an increased lossless rate. In fact, as we can see, the lossless rate is actually improved as well.

Additional commentary. From the results presented above, it is clear that the coding performance of the proposed method is vastly superior to that of the IT scheme. Here, we briefly comment on some of the reasons for this better performance. The IT method encodes all sample-position and sample-value data for a particular node at once before proceeding to code information for another node. In contrast, our proposed method does not code all sample-value data for a particular node together. By allowing sample-value data for different nodes to be interspersed, it is possible to better concentrate the information that most helps to reduce the reconstruction error earlier in the coded bitstream. This leads to our proposed method yielding superior progressive performance at intermediate rates. Another reason for the performance difference can be attributed to the different manner in which the sample-value data is represented in the tree. Due to this difference, in the case of a node with more than one child, the IT method must code an additional small integer (of 1 or 2 bits) relative to the proposed method. This helps to contribute to the proposed method achieving a lower lossless rate.

VI. Conclusions

In this paper, we have proposed a new progressive lossy-to-lossless coding method for arbitrarily-sampled image data. Through experimental results, the progressive coding efficiency of our proposed method at intermediate rates was shown to be vastly superior to that of the state-of-the-art IT scheme, yielding image reconstructions with much higher PSNR (sometimes by several dB). Furthermore, our proposed method was also demonstrated to yield slightly lower lossless rates. Our proposed coding method can benefit the many applications that utilize nonuniformly sampled images, by allowing such data to be more efficiently stored and communicated.

References

[1] I. Amidror, "Scattered data interpolation methods for electronic imaging systems: a survey," *Journal of Electronic Imaging*, vol. 11, no. 2, pp. 157–176, Apr. 2002.

[2] M. D. Adams, "A highly-effective incremental/decremental Delaunay mesh-generation strategy for image representation," *Signal Processing*, vol. 93, no. 4, pp. 749–764, Apr. 2013.

[3] P. Li and M. D. Adams, "A tuned mesh-generation strategy for image representation based on data-dependent triangulation," *IEEE Trans. on Image Processing*, vol. 22, no. 5, pp. 2004–2018, May 2013.

[4] L. Demaret and A. Iske, "Scattered data coding in digital image compression," in *Curve and Surface Fitting: Saint-Malo 2002*. Brentwood, TN, USA: Nashboro Press, 2003, pp. 107–117.

[5] M. D. Adams, "An efficient progressive coding method for arbitrarily-sampled image data," *IEEE Signal Processing Letters*, vol. 15, pp. 629–632, 2008.

[6] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[7] "USC-SIPI image database," 2011. [Online]. Available: http://sipi.usc.edu/database

[8] "JPEG-2000 test images," ISO/IEC JTC 1/SC 29/WG 1 N 545, Jul. 1997.