An Efficient Progressive Coding Method for Arbitrarily-Sampled Image Data

Michael D. Adams, *Member, IEEE*

***Abstract*—A simple highly-effective method for progressive lossy-to-lossless coding of arbitrarily-sampled image data is proposed. This scheme is based on a recursive quadtree partitioning of the image domain along with an iterative sample-value averaging process. The proposed method is shown to offer much better progressive coding performance than a previously-proposed state-of-the-art coding method.**

***Index Terms*—image coding, meshes, progressive lossy/lossless.**

I. Introduction

RECENTLY, there has been a growing interest in image-coding methods that better exploit the geometric structure inherent in images. A recurring theme in many of these methods (e.g., [1]–[3]) is the use of arbitrary sampling (i.e., sampling at an arbitrary subset of points from a lattice). In this context, the need to efficiently code arbitrarily-sampled image data arises. One particularly effective solution to this problem has been proposed in [4]. Since progressive coding functionality is desirable in many applications, a coding method's progressive performance is also frequently of interest. In this paper, we present a new data structure for representing arbitrarily-sampled image data, and propose a simple yet effective progressive lossy-to-lossless scheme for coding the information in this data structure. Although a number of methods other than [4] have been proposed to date for the coding of arbitrarily-sampled image data (e.g., [3]), these other methods do not provide fully progressive lossy-to-lossless functionality.

The remainder of this paper is structured as follows. Section II introduces some basic notation used herein. Then, in Section III, a new data structure for representing arbitrarily-sampled image data is presented, and in Section IV an efficient method for coding the information contained in this data structure is proposed. Section V evaluates the performance of our coding method, showing it to compare quite favorably with a previously-proposed state-of-the-art scheme. Finally, Section VI concludes the paper with a summary of our results.

II. Notation and Terminology

Before proceeding further, a brief digression is in order concerning the notation used herein. The sets of integers and real numbers are denoted as $\mathbb{Z}$ and $\mathbb{R}$, respectively. For $x \in \mathbb{R}$, $\lfloor x \rfloor$ denotes the largest integer not greater than x and $\lceil x \rceil$ denotes the smallest integer not less than x. For $m, n \in \mathbb{Z}$, we define the function $\operatorname{mod}(m, n) = m - n \lfloor m/n \rfloor$ (i.e., n divides into m with remainder $\operatorname{mod}(m, n)$). For $a, b \in \mathbb{Z}$, the notation $[a, b]$ and $[a, b)$ denote the subsets of $\mathbb{Z}$ given by $\{x \in \mathbb{Z} : a \leq x \leq b\}$ and $\{x \in \mathbb{Z} : a \leq x < b\}$, respectively. The unit-step sequence is denoted as u.

Manuscript received May 9, 2008; revised June 30, 2008. This work was supported by the Natural Sciences and Engineering Research Council of Canada. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Konstantinos Plataniotis.

The author is with the Dept. of Elec. and Comp. Eng., University of Victoria, Victoria, BC, V8W 3P6, Canada (e-mail: mdadams@ece.uvic.ca).

Digital Object Identifier XXXXXXXXXX

III. Image Tree

An image is an integer-valued function f defined for points $(x, y) \in \mathbb{Z}^2$ in the image domain D, with x, y, and $z = f[x, y]$ corresponding to horizontal position, vertical position, and intensity, respectively. Without loss of generality, we assume D to be rectangular and the sample values of f to be unsigned. In what follows, let W and H denote the width and height of D so that $D = [0, W) \times [0, H)$, and let P denote the number of bits per sample. Suppose now that we have f sampled at an arbitrary set of points in D. We would like a means to represent such a dataset that is both efficient and well suited for coding purposes. In what follows, we propose a data structure, called an image tree, that fulfills this desire.

An image tree is associated with an L-level quadtree partitioning of D into rectangular regions called cells, where $L = \lceil \log_2 \max(W, H) \rceil + 1$. The root cell of the quadtree is chosen as D, and the remaining cells are formed by recursively splitting the root cell to a maximum of L levels. In particular, a given cell $C = [x_0, x_1) \times [y_0, y_1)$ is split in (approximately) half in each of the horizontal and vertical directions to yield the four child cells C_0, C_1, C_2, and C_3 given respectively by

$$\begin{gathered} [x_0, x_\mathsf{m}) \times [y_0, y_\mathsf{m}), \quad [x_\mathsf{m}, x_1) \times [y_0, y_\mathsf{m}), \\ [x_0, x_\mathsf{m}) \times [y_\mathsf{m}, y_1), \quad \text{and} \quad [x_\mathsf{m}, x_1) \times [y_\mathsf{m}, y_1), \end{gathered} \tag{1}$$

where $x_\mathsf{m} = \left\lfloor \frac{1}{2}(x_0 + x_1 + 1) \right\rfloor$ and $y_\mathsf{m} = \left\lfloor \frac{1}{2}(y_0 + y_1 + 1) \right\rfloor$. As a matter of terminology, the area of a cell is defined as the number of lattice points (from $\mathbb{Z}^2$) it contains. A cell is said to be empty if it contains no sample points (from the arbitrarily-sampled dataset), and is said to be degenerate if it has zero area. Note that, by definition, a degenerate cell is always empty. Two nondegenerate cells are called neighbours if they are located at the same level in the quadtree and are 8-connected (i.e., immediately adjacent in the horizontal, vertical, or diagonal directions) in D.

As the name suggests, an image tree is a tree-based data structure. Each node in the tree is associated with: 1) a nonempty cell from the quadtree partitioning of D; and 2) a representative sample value (i.e., z value) for all of the sample points contained in this cell. Since each node is always associated with a nonempty cell, in the (uninteresting) degenerate case where the number of sample points is zero, the tree is empty (i.e., contains no nodes). Suppose now that we have at least one sample point. In this case, the root node is associated with a cell consisting of D. Then, the remainder of the tree hierarchy is constructed by recursively splitting the root node. More specifically, this node splitting process works

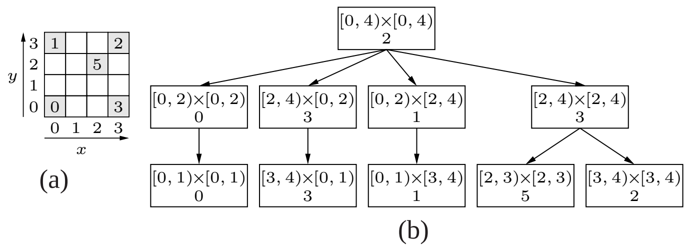


Fig. 1. Image tree example. (a) Image dataset; and (b) its corresponding image tree.

as follows. Let Q denote a node to be considered for splitting and let C denote its corresponding cell. If C has unit area, we are done (since a node having a cell of unit area is not split). Otherwise, we split C according to (1) to generate the child cells $\{C_i\}_{i=0}^{3}$. For each nonempty cell C' in $\{C_i\}_{i=0}^{3}$, a new child node is added to Q with the corresponding cell C'. Thus, the maximum number of children that Q can possess is equal to the number of nondegenerate cells in $\{C_i\}_{i=0}^{3}$. Lastly, each of the newly added child nodes is (recursively) split. Excluding leaf nodes, each node in the tree can have a maximum of either two or four children, with the first case arising only if W or H is not an integer power of two, or $W \neq H$.

The leaf nodes of the tree have a one-to-one correspondence with sample points. So, for each of these nodes, the representative sample value is simply chosen as the sample value for the corresponding sample point. For each of the remaining (i.e., nonleaf) nodes, the representative sample value is computed using a simple iterative averaging process. In particular, the representative sample value z for a nonleaf node with N children having representative sample values $\{z_i\}_{i=0}^{N-1}$ is given by

$$z = \left\lfloor \tfrac{1}{N} \left(\textstyle\sum_{i=0}^{N-1} z_i + \beta(N) \right) \right\rfloor, \tag{2}$$

where $\beta(n) = \lfloor n/2 \rfloor \, u[n-3]$. In the preceding equation, the additive bias $\beta(N)$ simply serves to reduce rounding error. Also, it is important to note that z has the same dynamic range as the $\{z_i\}_{i=0}^{N-1}$. Consequently, the representative sample value for each node in the tree can be represented using a P-bit (unsigned) integer.

To further assist in visualizing the structure of an image tree, we now provide a simple example. Fig. 1(a) depicts a simple image dataset with five sample points (shaded grid points) taken from the image domain $[0,4) \times [0,4)$. The corresponding image tree is shown in Fig. 1(b), where the cell and representative sample value for each node are indicated.

Observe that the image dataset (i.e., sample points and corresponding sample values) can be losslessly reconstructed from the information in the leaf nodes of the image tree. Furthermore, we can also generate approximations of the dataset from pruned versions of the tree. To do this, we generate one sample point and corresponding sample value for each leaf node in the pruned tree. In the case that a leaf node has a cell with area greater than one, the corresponding sample point is taken to be the (approximate) centroid of the cell. Thus, by coding the information in an image tree using a top-down traversal of the nodes in the tree, we can obtain a progressive encoding of the image dataset.

IV. Coding of Image Tree

Having introduced the image-tree data structure, we now propose a method for efficiently coding the information in such a tree. Our scheme employs a context-adaptive binary arithmetic coder [5]. In what follows, we describe only the encoding process since, from it, the decoding process can be easily deduced.

As suggested previously, we can construct a progressive encoding of an image tree by coding the information in the nodes using a top-down traversal of the tree. Clearly, many legal traversal orders are possible. The only constraint is that the information for a given node cannot be coded before the information for its parent node. To allow for flexibility in the traversal order, a heap-based priority queue, called the work queue, is employed during the coding process. The work queue holds all of the nodes that have been coded (i.e., have had their cell and representative sample value coded) but have not yet had their child information coded.

To commence the encoding process, a small fixed-length header is output containing the values of W, H, and P. Also, the work queue is cleared. If the tree is not empty, we encode the representative sample value for the root node as a P-bit (unsigned) integer using P bits without arithmetic coding, and insert the root node in the work queue. Since all subsequent data is arithmetically coded, the arithmetic coding engine is initialized at this point. Then, we loop, processing nodes in the work queue, until the work queue is empty. In each iteration, we first remove the next (i.e., highest priority) node from the work queue. Then, if the node is a nonleaf, we code the node's child configuration (i.e., the number of child nodes and the cells with which the child nodes are associated). Next, we code the representative sample value for each of the child nodes. The preceding two coding processes are described in detail later in this section. Lastly, for each of the child nodes, we compute its priority (for queue insertion) and insert it in the work queue.

By appropriately choosing the function used to compute node priority, we can effectively force the tree to be traversed in any (legal) order, thus controlling the order in which tree information is coded. Although many different traversal orders are possible, in this work we only employ breadth-first traversal (i.e., one tree level at a time from top to bottom) with raster-scan ordering by cell centroid within each tree level. Although this traversal order is not guaranteed to be optimal in any sense, it was experimentally found to perform well for a wide variety of datasets (compared to other simple breadth-first and depth-first orders). The consideration of other traversal orders is left as a topic for future investigation.

Arithmetic Coding. Frequently, when using a binary arithmetic coder, the need arises to code a bit with a fixed uniform probability distribution. In fact, most popular arithmetic coders provide what is called a "bypass mode" for accomplishing just this. Henceforth, whenever we speak of coding a bit in bypass mode, we are simply referring to the arithmetic coding of a bit with a fixed uniform distribution.

Since the arithmetic coder employed by our method is binary, we must specify a binarization scheme for each type of nonbinary symbol to be coded. There are three such types of symbols to be considered. The first type of nonbinary symbol is a ternary value with a fixed uniform distribution. Here, we require one additional context c_{third} in which the value of one has a fixed probability of $\frac{1}{3}$. The ternary value $n \in [0,2]$ is then binarized as a bit with value $\lfloor n/2 \rfloor$ coded using context c_{third} followed, if $\lfloor n/2 \rfloor = 0$, by a bit with value $\operatorname{mod}(n,2)$ coded in bypass mode. The second type of nonbinary symbol used is a hexary value with a fixed uniform distribution. The hexary value $n \in [0,5]$ is binarized as a bit with value $\lfloor n/3 \rfloor$ coded in bypass mode, followed by a ternary symbol with value $\operatorname{mod}(n,3)$.

The third type of nonbinary symbol is an n-bit unsigned integer with an adaptive nonuniform distribution. For such a symbol, we define a family of binarizations parameterized by an integer f, where $f \in [1,n]$. For convenience, we denote such a binarization as $\operatorname{UI}(n,f)$. The binarization of the value v with bits $\{b_i\}_{i=0}^{n-1}$, where $v = \sum_{i=0}^{n-1} b_i 2^i$ is performed as follows. For each bit position k from $n-1$ down to 0, we perform the following two steps: 1) Code the kth bit b_k using context c, where $c \in [0, 2^f + n - f - 1)$ and

$$c = \begin{cases} 2^{f-1} - 1 + \sum_{i \in [k+1,f)} (2b_i - 1)2^{i-1} & k \in [0, f-1] \\ 2^f - f + k - 1 & k \in [f, n-1]. \end{cases}$$

2) If $b_k = 1$ and $k \geq f$, we code each of the remaining bits $\{b_i\}_{i \in [0,k)}$ in bypass mode, and terminate the loop early. Effectively, the preceding process allows for each symbol value in the range $[0, 2^f)$ to be assigned a distinct probability, with the remaining values (if any) partitioned into the ranges $[2^i, 2^{i+1})$ for $i \in [f, n)$, where the values within each range are equiprobable. This scheme can accommodate the coding of integers with a variety of probability distributions, and is most flexible when $f = n$ in which case a distinct probability can be used for each symbol.

Coding of Child Configuration. With the preceding binarization schemes introduced, we are now in a position to present the remaining details of our coding method. To begin, we describe how the child configuration is coded for a given node Q. Let M denote the maximum possible number of children that Q can possess (which is determined by the node splitting process as described earlier), and let N denote the number of children that Q actually has. To convey the child configuration for Q, we choose to code N followed by an indication of which N of the M possible child nodes are present.

First, we consider the coding of the quantity N. Let C denote the cell of Q. Let A denote the set of all (nondegenerate) neighbour cells of C in the quadtree partitioning of D, and let E denote the number of cells in A that are, from information previously coded, known to be empty. Let $\{K_i\}_{i \in [0,K)}$ denote the set of all previously coded nodes that are associated with (nonempty) cells in A, with N_i and M_i denoting the number of children and maximum possible number of children of the node K_i. For use in context selection, we compute the integer quantity p as

$$p = \begin{cases} \operatorname{clip}\left(\left\lfloor \frac{M}{K+E} \sum_{i \in [0,K)} \frac{N_i}{M_i} + \frac{1}{2} \right\rfloor, 1, M \right) & K + E > 0 \\ 0 & K + E = 0, \end{cases}$$

where $\operatorname{clip}(x,a,b) = \min(\max(x,a),b)$. Then, we code $N - 1$, where $N - 1 \in [0, M)$ is an η-bit integer and $\eta = \log_2 M$. This coding is done using $\operatorname{UI}(\eta,\eta)$ binarization, conditioned on M, p, and the level in the tree where Q resides. Since sample points often cluster together, neighbouring nodes (i.e., nodes with neighbouring cells) tend to have a similar number of children. Thus, in the preceding process, we use information about the nodes neighbouring Q (or the lack of such nodes) in order to estimate N. In particular, the quantity p corresponds to an estimate of $N \in [1, M]$ with the special out-of-range value of 0 assigned in the (typically) infrequent case that insufficient information is available for making a good estimate.

Next, we must encode which N of M possible children are present. In general, there are $\binom{M}{N}$ possible child configurations. Recall that $M \in \{2,4\}$. If $M = 2$, we have two cases to consider, namely $N \in \{1,2\}$. If $N = 1$, one of two possible configurations is coded using a single bit in bypass mode. If $N = 2$, only one configuration is possible, and no information need be coded. If $M = 4$, we have four cases to consider, namely $N \in [1,4]$. If $N \in \{1,3\}$, one of $\binom{4}{1} = \binom{4}{3} = 4$ possible configurations is coded using two bits, each in bypass mode. If $N = 2$, one of $\binom{4}{2} = 6$ possible configurations is coded using a single hexary symbol. If $N = 4$, only one configuration is possible, and no information need be coded.

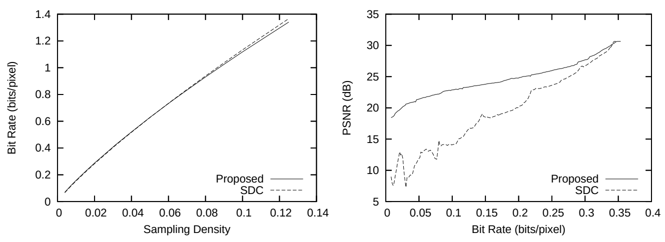
Coding of Representative Sample Values. Now, let us consider the coding of the representative sample values for the children of a given node Q. Let M and N respectively denote the maximum possible and actual number of children for Q. Let z denote the representative sample value of Q and z_i denote the representative sample value for the ith child of Q. If $N = 1$, we have (from (2)) that $z_0 = z$, and no information need be coded. Suppose now that $N \geq 2$. For k from 0 to $N - 1$, we code each z_k as follows. Rather than coding the value of z_k directly, we code its difference from some reference value. Based on the information coded so far, we compute s_0 and s_1, the minimum and maximum possible values, respectively, for the sum $\sum_{i=k}^{N-1} z_i$ as $s_0 = Nz - \sum_{i \in [0,k)} z_i - \beta(N)$ and $s_1 = s_0 + N - 1$. There are two cases to consider: 1) $k = N - 1$ and 2) $k < N - 1$. If $k = N - 1$, we have $z_k \in [s_0, s_1]$ and we code the residual $r = z_k - s_0$, where $r \in [0, N)$. For the case that N is 2, 3, or 4, r is respectively coded as one bit in bypass mode, a single ternary symbol, or two bits in bypass mode. If $k < N - 1$, we predict z_k as

$$p = \begin{cases} \left\lfloor \frac{1}{2(N-k)} (s_0 + s_1 + \beta(2[N-k])) \right\rfloor & k \in [1, N-1) \\ z & k = 0. \end{cases}$$

We code the residual $r = z_k - p$ as follows, where $r \in [-(2^P - 1), 2^P - 1]$. The P-bit integer $|r|$ is coded using $\operatorname{UI}(P, \min(P,4))$ binarization, conditioned on the node's level in the tree. Next, if $|r| \neq 0$, the sign of r is coded as single bit in bypass mode. The residual r tends to be close to zero, since the remaining unknown $\{z_i\}_{i=k}^{N-1}$ are frequently close to their average value (for which p is an estimate).

V. Experimental Results

To demonstrate the effectiveness of our proposed coding method, we compare its coding performance to that of the scattered data coding (SDC) scheme [4] (which is used, for example, in [2]) with a breadth-first traversal order and the minor enhancements described in [1]. As originally proposed, the SDC method does not employ arithmetic coding. Therefore, to allow a fairer comparison, we applied arithmetic coding to the symbol stream produced by the SDC method. This typically reduces the lossless bit rate achieved with the SDC method by 2 to 5%. For test data, we employed several lattice-sampled images, including the well-known lena image. To generate arbitrarily-sampled datasets from (lattice-sampled) images and vice versa, we used the MGH mesh-generation



(a) (b)

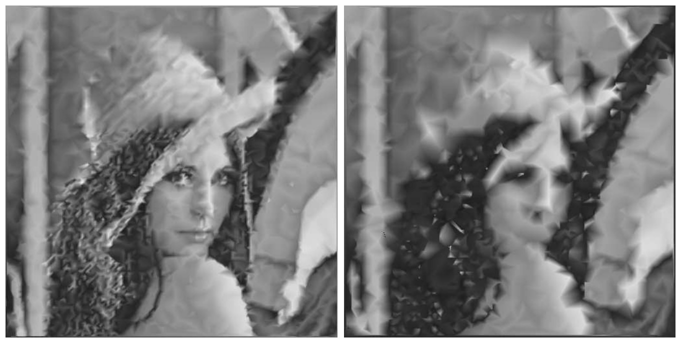
Fig. 2. Coding results for the lena image obtained using the proposed and SDC methods. (a) Lossless coding performance. (b) Progressive coding performance (with a dataset sampling density of 0.025).

method and corresponding triangulation-based interpolation scheme from [1].

First, we consider lossless coding performance. For several (lattice-sampled) test images, arbitrarily-sampled datasets with different numbers of sample points were generated. Then, each dataset was losslessly coded using the proposed and SDC methods, and the final bit rates measured. A representative subset of the results, namely for the lena image, is shown in Fig. 2(a). From these results, we can see that our proposed method performs comparably to the SDC method at low sampling densities, and better at higher sampling densities. Although our interest herein is with progressive performance, these results are important in that they show that any improvement in progressive performance that our method may offer does not come at any significant cost in terms of the final lossless bit rate.

Now, we consider progressive coding performance (i.e., decoding in a lossy manner to various intermediate rates). For several images, arbitrarily-sampled datasets corresponding to various sampling densities were generated. Using each of the proposed and SDC methods, each dataset was coded once losslessly, and then decoded at many intermediate rates. In each case, the decoded dataset was interpolated to produce a lattice-sampled image and the peak-signal-to-noise ratio (PSNR) relative to the original (lattice-sampled) image was computed. A representative subset of the results, namely for the lena image, is shown in Fig. 2(b). In the graph, the maximum PSNR attained corresponds to lossless reconstruction of the arbitrarily-sampled dataset. Clearly, the proposed method outperforms the SDC method at lower rates by a very large margin (often by several dB). Only when the arbitrarily-sampled dataset is very nearly losslessly decoded does the SDC method sometimes have slightly better performance. To show that the PSNR results correlate well with subjective quality, two of the obtained intermediate image reconstructions are shown in Fig. 3. Clearly, the reconstruction produced by our proposed method is of much better quality than the one generated by the SDC scheme.

In terms of both computational and memory complexity, the proposed method is also superior to the SDC scheme. The execution times for the encoding and decoding algorithms were both typically found to be 2 to 4 times greater for the SDC method than the proposed method. Furthermore, the SDC method was also observed to typically require over 10 times more memory than the proposed method. These complexity characteristics can be explained as follows. The proposed method is based on a quadtree partitioning of a 2-D space of size WH, while the SDC method is based on an octree partitioning of a 3-D space of size $2^P WH$. Therefore, the



(a) (b)

Fig. 3. Intermediate image reconstructions for the lena image (with a dataset sampling density of 0.025) obtained at 50:1 compression using the (a) proposed (23.97 dB) and (b) SDC (18.41 dB) methods.

SDC method must partition a much larger space, resulting in a tree with a much greater number of nodes (typically, 6 to 7 times more). Furthermore, in the SDC case, the nodes are also larger, since each node must have pointers for eight children instead of four. Since memory usage is dominated by the tree data structure in both methods, the proposed scheme requires less memory. Since both methods must process every tree node during encoding or decoding, the much greater node count for the SDC method leads to a longer execution time relative to the proposed method.

The relatively poor progressive performance of the SDC method is due to the fact that it represents each sample point (x, y) and its corresponding sample value z as a point (x, y, z) in 3-D space. Unfortunately, as a result of this, at intermediate stages of decoding, it is possible to obtain multiple points (in 3-D) with the same x and same y coordinates but distinct z coordinates, which corresponds to a single sample point having multiple distinct sample values. Such ambiguities ultimately lead to degraded coding performance, and also cause the unusual oscillatory behavior in the rate-distortion curve for the SDC method in Fig. 2(b) at lower rates. Although various strategies can be used in an attempt to reduce the degradation caused by these ambiguities, the improvement offered by such strategies is fundamentally limited.

VI. Conclusions

In this paper, we have proposed a new method for progressive lossy-to-lossless coding of arbitrarily-sampled image data. Our method is conceptually simple, flexible (allowing for different progression orders), and was shown to yield much better progressive performance than a previously-proposed state-of-the-art coding technique at lower computational and memory costs. By using our proposed method, one can hope to develop improved mesh-based image coders in the future.

References

[1] M. D. Adams, “An evaluation of several mesh-generation methods using a simple mesh-based image coder,” in *Proc. of IEEE ICIP*, San Diego, CA, USA, Oct. 2008, to appear.

[2] L. Demaret and A. Iske, “Adaptive image approximation by linear splines over locally optimal Delaunay triangulations,” *IEEE Sig. Proc. Letters*, vol. 13, no. 5, pp. 281–284, May 2006.

[3] L. Demaret, N. Dyn, and A. Iske, “Image compression by linear splines over adaptive triangulations,” *Sig. Proc.*, vol. 86, pp. 1604–1616, 2006.

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