PROGRESSIVE LOSSY-TO-LOSSLESS CODING OF ARBITRARILY-SAMPLED IMAGE DATA USING THE MODIFIED SCATTERED DATA CODING METHOD

Michael D. Adams

Dept. of Elec. and Comp. Eng., University of Victoria, Victoria, BC, V8W 3P6, Canada

ABSTRACT

In earlier work, Demaret and Iske proposed the scattered data coding (SDC) method for (single-rate) coding of arbitrarily-sampled image data. In this paper, several modifications to the SDC method are proposed in order to remove some limitations of the original scheme, improve coding efficiency, and add a progressive lossy-to-lossless coding capability. Through experimental results, the proposed method is shown to yield a significant improvement in coding efficiency (relative to the original SDC method) as well as provide an efficient progressive lossy-to-lossless coding capability.

Index Terms— image coding, meshes, progressive coding

1. INTRODUCTION

In recent years, there has been a growing interest in image coding methods that better exploit the nonstationarity and geometric properties of images. Many of these methods (e.g., [1, 2, 3, 4]) are based on the idea of employing arbitrary sampling (i.e., sampling at an arbitrary subset of points from a lattice). In this context, the need to code arbitrarily-sampled image data arises. One highly effective scheme for the coding of such data is the scattered data coding (SDC) method proposed by Demaret and Iske [2]. Unfortunately, the SDC coder, as originally proposed in [2], has some significant limitations. In particular, the width and height of the image to be coded are assumed to be equal and integer powers of two. Also, images with high-precision sample data cannot be handled (more specifically, images where the number of bits per sample is greater than the number of bits needed to represent the image width/height). This is problematic for various types of medical/scientific imagery. Lastly, the SDC coder, as originally proposed, did not address the issue of progressive coding functionality. In many applications, progressive coding functionality is beneficial or even required. In this paper, we propose a modified version of the SDC coder that eliminates the above limitations, offers improved coding efficiency, and most importantly provides an efficient progressive lossy-to-lossless coding capability.

The remainder of this paper is structured as follows. First, Section 2 introduces some of the basic notation used herein. Section 3 provides some background information related to the SDC coder. Then, Section 4 proposes a modified version of the SDC coder, and explains how it can be used for progressive coding. Section 5 explores the impact that various parameters of our modified coder have on coding efficiency. This section also compares the coding performance of the proposed method to the SDC method, showing the proposed scheme to yield better performance. Finally, Section 6 concludes this paper with a summary of our work.

2. 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 $a, b \in \mathbb{Z}$, 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 $\{x \in \mathbb{Z} : a \leq x < b\}$.

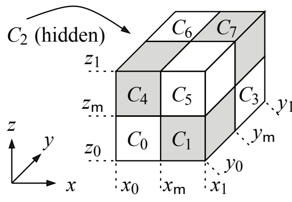
3. BACKGROUND

A grayscale image is a function f defined for points (x, y) in the image domain $D \in \mathbb{Z}^2$, where x, y, and $z = f[x, y]$ correspond to horizontal position, vertical position, and intensity, respectively. Suppose that a dataset has been generated by sampling f at an arbitrary subset of points in D. We would like a convenient data structure for representing such a dataset as well as an algorithm for efficiently coding the information in this structure. In what follows, let W and H denote the image width and height, respectively, and assume that the sample values are unsigned integers in $[0, D)$. The SDC method [2] views the above dataset as a collection of points in the 3-dimensional (3-D) region $\mathcal{I} = [0, W) \times [0, H) \times [0, D)$. In particular, each **sample position** (x_i, y_i) and corresponding **sample value** z_i is represented by a **sample point** (x_i, y_i, z_i) in $\mathcal{I}$. In short, the SDC method uses a tree-based data structure to represent this collection of sample points, and provides an algorithm for efficiently coding the information in this structure. In what follows, we propose a modified version of the SDC method, called the modified SDC (MSDC) method, that overcomes some limitations of the SDC scheme and further improves performance. Due to the similarities between the SDC and MSDC methods, we will simply introduce our MSDC method in full, and briefly comment on how it differs from the SDC scheme. For a complete description of the SDC method, the reader is referred to [2].

4. MODIFIED SDC METHOD

Image Tree. In a similar (but not identical) fashion as the SDC approach, the MSDC method uses a tree-based data structure, called an image tree, to represent the set of sample points in $\mathcal{I}$. An image tree is associated with an L-level octree partitioning of $\mathcal{I}$ into (3-D) hyperrectangular regions called cells, where $L = \lceil \log_2 \max\{W, H, D\} \rceil + 1$. The root cell of the octree is chosen as $\mathcal{I}$, and the remaining cells are generated by recursively splitting the root cell. In particular, a given cell $C = [x_0, x_1) \times [y_0, y_1) \times [z_0, z_1)$ is split at its approximate midpoints in the x, y and z directions to yield eight new child cells $\{C_i\}_{i \in [0,8)}$ as shown in Fig 1 (e.g., $C_0 = [x_0, x_\mathsf{m}) \times [y_0, y_\mathsf{m}) \times [z_0, z_\mathsf{m})$). For convenience, the size of a cell is expressed using the notation $x \times y \times z$, where x, y, and z are the extents of the cell in the x, y, and z directions, respectively. As a matter of terminology, the volume of a cell is defined as the number of lattice points (from $\mathbb{Z}^3$) it contains. A cell is said to be degenerate if it has zero volume, empty if it contains no sample points, and full if the number of contained sample points equals the cell volume. Also, a cell is said to be atomic if it is neither empty nor full, and its volume is four or less.

This work was supported by the Natural Sciences and Engineering Research Council of Canada.



$$p_{\mathsf{m}} = (x_{\mathsf{m}}, y_{\mathsf{m}}, z_{\mathsf{m}}) \quad x_{\mathsf{m}} = \lfloor \tfrac{x_0+x_1+1}{2} \rfloor$$
$$y_{\mathsf{m}} = \lfloor \tfrac{y_0+y_1+1}{2} \rfloor \quad z_{\mathsf{m}} = \lfloor \tfrac{z_0+z_1+1}{2} \rfloor$$
$$C_{0246} = C_0 \cup C_2 \cup C_4 \cup C_6$$
$$C_{1357} = C_1 \cup C_3 \cup C_5 \cup C_7$$
$$C_{04} = C_0 \cup C_4 \quad C_{26} = C_2 \cup C_6$$
$$C_{15} = C_1 \cup C_5 \quad C_{37} = C_3 \cup C_7$$

Fig. 1. Cell splitting.

As the name suggests, an image tree is a tree-based data structure. Each node in the tree is associated with: 1) a cell C in the octree partitioning of I; 2) a count indicating the number of sample points contained in C; and 3) if C is atomic, an indication of which lattice points in C are occupied by sample points. The root node of the tree is associated with the root cell of the octree partitioning of I, and the remainder of the tree is constructed by recursively splitting the root node. This node splitting process works as follows. Let Q denote a node to be considered for splitting, and C its corresponding cell. If C is empty, full, or atomic, we do nothing (since such cells are not split). Otherwise, we split C as shown earlier in Fig. 1 to produce the child cells $\{C_i\}_{i\in[0,8)}$. For each nondegenerate cell C' in $\{C_i\}_{i\in[0,8)}$, a new child node is added to Q with a corresponding cell C'. Thus, the maximum number of children that Q can possess is eight. Lastly, each of newly added child nodes is (recursively) split.

Now, we briefly outline the differences between the image-tree data structure used in our MSDC method (described above) and that used in the SDC method. The key difference is the way in which the cell splitting point p_{m} is chosen in Fig. 1. The SDC coder can simply choose p_{m} as $(\frac{x_0+x_1}{2}, \frac{y_0+y_1}{2}, \frac{z_0+z_1}{2})$, since this quantity is always an integer lattice point as a consequence of the restrictions that the coder imposes on W, H, and D. Since our MSDC coder imposes no such restrictions, it must employ the more general formula for p_{m} given in Fig. 1 (which involves a rounding operation). In the SDC method, all nonleaf nodes must have eight children (since degenerate cells cannot occur during node splitting), while in our MSDC method, degenerate cells can arise, and consequently nonleaf nodes can have less than eight children. In the SDC method, an atomic cell always has size $2\times 2\times 1$. In our scheme, we have extended the definition of an atomic cell to also include several other cell sizes (which are identified later).

Due to the manner in which an image tree is constructed, every sample point from the original arbitrarily-sampled image dataset is associated with a leaf node in the tree. Consequently, the dataset can be losslessly reconstructed from the leaf nodes. Now, we make a crucial observation, namely, that approximations of the original dataset can be obtained from pruned versions of the tree. In particular, given a pruned tree, we can generate the sample points for such an approximation as follows. For each leaf node Q with corresponding cell C, if C is nonempty, do the following: 1) if all sample points in C are known (i.e., C is full or atomic), generate one sample point for each known sample point in C; 2) otherwise, generate a single sample point corresponding to the (approximate) centroid of C (i.e., $(\lfloor\frac{x_0+x_1}{2}\rfloor, \lfloor\frac{y_0+y_1}{2}\rfloor, \lfloor\frac{z_0+z_1}{2}\rfloor)$ in Fig. 1). Since pruned versions of an image tree are associated with approximations of the original dataset, we can obtain a progressive encoding of the dataset by coding the information in the tree using a top-down traversal of its nodes.

Coding of Image Tree. Having introduced the image tree, we now explain how the information in such a tree is coded. Unlike the SDC method (which employs Huffman coding), our scheme employs multisymbol arithmetic coding [5]. In what follows, whenever we refer to coding an integer $i \in [0,m)$ as an m-ary symbol, we simply mean that i is arithmetically coded using a fixed uniform probability distribution. For convenience, we define the case of $m = 1$ (i.e., coding a symbol from the alphabet with the single symbol 0) to be a no-op (i.e., an operation that does nothing). Below, we describe only the encoding process, since from it, the decoding process can be easily deduced.

As explained earlier, we can generate a progressive encoding of an image tree by coding information using a top-down traversal of the tree. The only constraint on traversal order is that a node cannot be visited before its parent. To facilitate the use of different traversal orders, we employ a (heap-based) priority queue called the **work queue**. The work queue simply serves to hold nodes awaiting processing by the encoder. The function used to compute node priority determines the traversal order. Any legal traversal order can be obtained with an appropriate choice of node-priority function.

To begin encoding, a small fixed-length header is written containing W, H, and D. Then, the work queue is cleared, and the arithmetic coding engine is initialized, as all subsequent symbols are arithmetically coded. For the root node, the number n of sample points in I (where $n \in [0, WH+1)$) is coded as a $(WH+1)$-ary symbol. The root node is inserted into the work queue. Then, we loop, processing nodes from the work queue until it is empty. In each iteration, we remove the next (i.e., highest priority) node from the work queue. If the node is empty or full, do nothing. Otherwise, encode the node's child information using a process to be described shortly. For each child node (of Q), compute its priority and insert it into the work queue.

Now, we describe how the child information for a node Q with cell C is coded. There are two cases to consider: 1) C is not atomic, and 2) C is atomic.

Case 1: If C is not atomic, we proceed as follows. Let $c(K)$ denote the number of sample points in a cell K. We need to split the cell C as shown in Fig. 1. For the time being, let us assume that no degenerate cells are produced during the splitting process. First, we split C in the x direction to yield two new subcells C_{0246} and C_{1357}. Then, we code $c(C_{0246})$ as a $[c(C)+1]$-ary symbol. Next, each of the two cells from the preceding step are split in the y direction to yield four subcells C_{04}, C_{26}, C_{15}, and C_{37}. Then, we code $c(C_{04})$ as a $[c(C_{0246})+1]$-ary symbol and $c(C_{15})$ as a $[c(C_{1357})+1]$-ary symbol. Next, each of the four cells from the preceding step are split in the z direction to yield the eight subcells $\{C_i\}_{i\in[0,8)}$. Then, we code $c(C_0)$ as a $[c(C_{04})+1]$-ary symbol, $c(C_1)$ as a $[c(C_{15})+1]$-ary symbol, $c(C_2)$ as a $[c(C_{26})+1]$-ary symbol, and $c(C_3)$ as a $[c(C_{37})+1]$-ary symbol. During the above split operations where a cell is split in two, we need only code the number η_0 of sample points that lie in one of the two new subcells, since the number η_1 of sample points in the other subcell can be deduced from η_0 and the number η of points in the cell being split (i.e., $\eta_1 = \eta - \eta_0$). During each of the above split operations, it is possible that one of the two new subcells produced may be degenerate. In this case, all of the sample points must lie in the nondegenerate subcell, and no information need be coded for the split operation.

Case 2: If C is atomic, the coding process proceeds as described below. In what follows, let v and c respectively denote the volume of C and number of sample points contained in C. The only possible cell sizes and corresponding count values are as follows: 1) $1\times 4\times 1$, $2\times 2\times 1$, and $4\times 1\times 1$, with $c\in[1,3]$; 2) $1\times 2\times 2$, $1\times 3\times 1$, $2\times 1\times 2$, and $3\times 1\times 1$, with $c\in[1,2]$; 3) $1\times 1\times 2$, $1\times 1\times 3$, $1\times 1\times 4$, $1\times 2\times 1$, and $2\times 1\times 1$, with $c = 1$. For some cell sizes, not all values of $c\in[1,v)$ are possible. For example, a cell with size

$1 \times 1 \times 3$ cannot have $c = 2$, since this would require a single sample position to have multiple distinct sample values. For nearly all of the cell sizes listed above, there are $n = \binom{v}{c}$ possible configurations of sample points within the cell. The only exceptions are the sizes $2 \times 1 \times 2$ and $1 \times 2 \times 2$. In these cases, we nominally have $\binom{4}{2} = 6$ possibilities. Two of these cases, however, cannot occur, since we cannot have two sample points with the same x and same y coordinates but distinct z coordinates. In these cases, only four possible sample-point configurations are possible. Let n be the total number of sample-point configurations for C. We code a value $i \in [0,n)$ as an n-ary symbol to indicate which of the n possible sample-point configurations has actually occurred.

Now, a few further comments are in order regarding the relationship between the MSDC and SDC methods. In the case that W, H, and D are all integer powers of two and $W = H$, the MSDC coder essentially degenerates into the SDC coder with the addition of arithmetic coding and progressive coding functionality. This degenerate situation, however, does not arise so frequently in practice, as (statistically speaking) most images are not square with integer-power-of-two dimensions. With regard to atomic (i.e., unsplittable) cells, the motivation behind their use is to prevent the splitting of many small cells. In the SDC method, atomic cells were introduced to improve coding efficiency. In our method, since arithmetic coding is employed, the use of atomic cells has little impact on coding efficiency. There is, however, an important reason for using atomic cells in our method, namely, by avoiding the splitting of small cells, the number of nodes in an image tree is significantly reduced (typically, by about 7 to 16%), resulting in considerable memory savings.

Sample-Value Ambiguity Problem. Although a progressive encoding of the information in an image tree can be generated (as discussed earlier), there is a fundamental problem associated with progressive coding that has yet to be addressed. This problem arises from the fact that the coder treats the sample position/value data as if it were truly 3-D data (i.e., as sample points in 3-D). In reality, however, this data is not completely arbitrary 3-D data. That is, for an arbitrary 3-D point set, it would be perfectly reasonable to have two points (x_0,y_0,z_0) and (x_1,y_1,z_1) with $x_0 = x_1$, $y_0 = y_1$, and $z_0 \neq z_1$. In the context of the coding problem at hand, however, this particular situation would correspond to a single sample position (x_0,y_0) having the two distinct sample values z_0 and z_1, which is clearly impossible. Unfortunately, due to the way in which the coder represents sample position/value data and generates approximations from pruned trees, it is possible, at intermediate stages of decoding, to obtain multiple distinct sample values for the same sample position. When such a situation arises, the decoder must resolve this ambiguity by selecting a single sample value to use for decoding purposes. For any given sample position, the probability of this ambiguity problem occurring is typically quite high initially (i.e., at very low rates) and then decreases with rate, often in a very oscillatory (i.e., non-monotonic) manner. Consequently, this ambiguity problem most significantly impacts the progressive coding performance at lower rates, often leading to rate-distortion curves with large oscillations at lower rates. To address the above problem and mitigate its effects, we consider four ambiguity-resolution methods. Suppose that the above ambiguity problem arises for the sample position (x_i,y_i) with the set $\mathcal{Z}$ of multiple distinct sample values. Let z_i denote the sample value to be used for decoding purposes. Our first method, called the **discard** method, simply discards the ambiguous sample point/value data altogether. Our second method, called the **nearest neighbour** method, finds the sample position closest to (x_i,y_i) that has only a single sample value z associated with it, and then chooses z_i as the value from $\mathcal{Z}$ that differs least in magnitude from z. The remaining two methods, known as the **mean** and **median** methods, choose z_i as the mean and median of $\mathcal{Z}$, respectively.

Progression Order. In our work, we consider six different progression orders. Recall that the progression order is determined solely by the node-priority function, which is used for inserting nodes into the work queue. Suppose that we want to compute the priority λ of the node Q with corresponding cell C. Let x, y, and z respectively denote the (integer-valued) x, y, and z coordinates of the (approximate) centroid of C. Let ℓ, v, and c respectively denote the level in the tree where Q resides (with the top being zero), the volume of C, and the number of sample points in C. The six progression orders and their corresponding node-priority functions are as follows: 1) **breadth first**: $\lambda = -x - Wy - WHz - WHD\ell$; 2) **depth first**: $\lambda = \ell$; 3) **count**: $\lambda = c$; 4) **density**: $\lambda = c/v$; 5) **sparsity**: $\lambda = v/c$; and 6) **deviation from half density (DFHD)**: $\lambda = |c/v - 1/2|$. Note that the breadth-first order corresponds to numbering the nodes according to a strict breadth-first ordering (i.e., one tree level at a time from top to bottom) with raster-scan ordering by cell centroid within each tree level. Also, observe that each of the above node-priority functions can be calculated using only previously coded information. Therefore, there is no need to explicitly code the progression order as side information.

5. EXPERIMENTAL RESULTS

Having introduced our MSDC coder, we now examine how the choices of sample-value ambiguity-resolution method and progression order affect our coder, and evaluate our coder's performance relative to the SDC coder. For test data, we employed more than a dozen lattice-sampled images, including the `lena` and `peppers` images from the well-known USC image database. To generate arbitrarily-sampled datasets from (lattice-sampled) images and vice versa, we used the MGH mesh-generation method and corresponding triangulation-based interpolation scheme from [1]. As a matter of terminology, in what follows, we refer to the reciprocal of compression ratio as the compression factor.

Sample-Value Ambiguity Resolution. Previously, we proposed four schemes for handling the sample-value ambiguity problem in our MSDC coder. We now present some results comparing the effectiveness of these schemes. Since the choice of ambiguity-resolution method only affects decoding, we need not consider lossless coding performance here. For several arbitrarily-sampled datasets generated from our test images, each dataset was coded once losslessly and then decoded in a progressive manner using each of the four ambiguity-resolution methods (i.e., discard, mean, median, and nearest neighbour). In each case, the peak-signal-to-noise ratio (PSNR) relative to the original (lattice-sampled) image was measured. A representative subset of the results, namely for the `lena` image, is shown in Fig. 2. In the graph, the maximum PSNR attained corresponds to lossless reconstruction of the arbitrarily-sampled dataset. Although these particular results were generated using the breadth-first progression order, similar results were also obtained for other progression orders. Recall that, due to the sample-value ambiguity problem, the rate-distortion curves for our MSDC coder can depart significantly from monotonic behavior. This behavior is clearly evident in the graph (i.e., Fig. 2). Comparing the results obtained with the various methods, we can clearly see that the median method yields the best results, often beating the other methods by more than 1 dB. Of the remaining methods, the nearest-neighbour scheme is next best, followed by the discard and mean methods.

Progression Order. Earlier, we proposed six different progression orders that can be used with our MSDC coder. Now, we com-

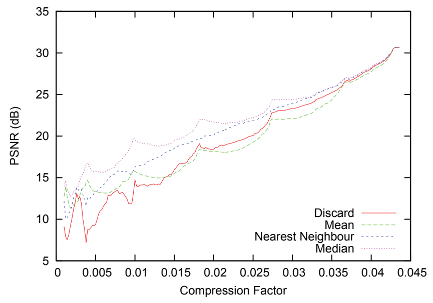


Fig. 2. Progressive coding results obtained using various sample-value ambiguity-resolution methods for the `lena` image with a dataset sampling density of 1/40.

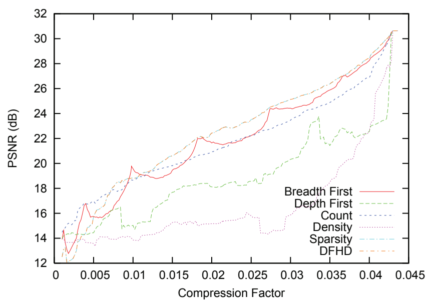


Fig. 3. Progressive coding results obtained using various progression orders for the `lena` image with a dataset sampling density of 1/40.

Table 1. Lossless coding results obtained using the MSDC and SDC methods for the (a) `lena` and (b) `peppers` images

(a)

Sampling Density	File Size (Bytes) MSDC	SDC	Relative Diff. (%)
1/40	11263	11734	4.0
1/30	14488	15114	4.1
1/20	20568	21498	4.3

(b)

Sampling Density	File Size (Bytes) MSDC	SDC	Relative Diff. (%)
1/40	11614	12087	3.9
1/30	14909	15532	4.0
1/20	21150	22053	4.1

pare these orders in terms of their coding efficiency. For several arbitrarily-sampled datasets generated from our test images, each dataset was losslessly coded and then decoded in a progressive manner using each of the six progression orders (i.e., breadth-first, depth-first, count, density, sparsity, and DFHD). In each case, the PSNR relative to the original (lattice-sampled) image was measured. A representative subset of the results, namely for the `lena` image, is shown in Fig. 3. In the graph, the maximum PSNR attained corresponds to lossless reconstruction of the arbitrarily-sampled dataset. Since changing the progression order does not affect the lossless rate, all of the progression orders achieve the same maximum PSNR at the same (i.e., lossless) rate. Here, we have chosen to present results that employ the median scheme for sample-value ambiguity resolution, as this method was previously found to work best. From the graph, it is clear that the DFHD and sparsity methods (which have very closely overlapping lines on the graph) outperform the other schemes by a significant margin (i.e., 0.5 to 1 dB or more). Although not discernible from the graph, a much closer comparison shows that the DFHD and sparsity schemes have a nonnegligible difference in performance only at very high (i.e., nearly lossless) rates, in which case the DFHD scheme performs slightly better. For this reason, the DFHD scheme is deemed to perform best, with the sparsity scheme taking second place. The breadth-first scheme is next most effective, followed by the count method. Since the DFHD and sparsity methods use a slightly more "irregular" traversal pattern than breadth-first scheme, they are less significantly impacted by the sample-value ambiguity problem, which contributes to better performance (and a more monotonic rate-distortion curve). The remaining methods all perform relatively poorly. As one might suspect, the depth-first scheme does not fare very well. The problem with progression orders that are mostly depth first is that they often result in very small areas of the image being refined, while other much larger areas with very high distortion remain unchanged. For this reason, the best performing schemes tend to be closer to a breadth-first order than a depth-first one.

MSDC Coder Versus SDC Coder. Finally, we compare the performance our MSDC coder to the SDC coder. Since the SDC coder does not provide progressive coding functionality, our comparison is limited to lossless coding performance. Furthermore, as the SDC coder can only handle images that are square with integer power of two width/height, our test images have been chosen to satisfy this constraint. For several (lattice-sampled) test images, arbitrary-sampled datasets with different sampling densities were generated. Then, each dataset was losslessly coded using the MSDC and SDC coders and the resulting (lossless) rates measured. A representative subset of the results is shown in Table 1. As these results demonstrate, our MSDC coder yields a lossless rate that is typically about 4% less than that obtained with the SDC coder. So, clearly, our MSDC coder outperforms the SDC coder in terms of lossless coding efficiency. Furthermore, our MSDC coder also offers additional functionality that is not provided by the SDC coder. First, our MSDC coder can handle images of arbitrary size (i.e., images need not be square or have integer power-of-two dimensions). Second, and perhaps more importantly, our MSDC coder provides progressive lossy-to-lossless coding functionality. The effectiveness of our progressive coding scheme is demonstrated by the results of Fig. 3.

6. CONCLUSIONS

In this paper, we have proposed the MSDC coder, a modified version of the SDC coder. Relative to the SDC coder, our MSDC coder has several key advantages, namely, it: 1) has support for images of arbitrary width, height, and sample precision, 2) offers better lossless coding performance, and 3) provides an efficient progressive lossy-to-lossless coding capability that can accommodate a wide range of progression orders. For applications where progressive transmission by fidelity is desired, we showed the DFHD progression order to be most effective. Moreover, we found that a simple median scheme was most effective at overcoming the sample-value ambiguity problem that arises during progressive decoding. By having developed an improved version of the highly effective SDC coder, we have made it possible to construct mesh-based image coders with higher coding efficiency.

7. REFERENCES

[1] M. D. Adams, "An evaluation of several mesh-generation methods using a simple mesh-based image coder," in *Proc. of IEEE ICIP*, Oct. 2008, pp. 1041–1044.

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

[3] 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.

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

[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.