Progressive Lossy-to-Lossless Coding of Arbitrarily-Sampled Image Data Using the Modified Scattered Data Coding Method

University of Victoria

Michael D. Adams

Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, V8W 3P6, Canada

mdadams@ece.uvic.ca

1. Introduction

To better exploit the nonstationary and geometric properties of images, many geometric-based image coders employ 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 [1]. Unfortunately, the SDC coder, as originally proposed in [1], 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, and the precision of the image sample data is assumed to be relatively small. These assumptions are not satisfied in many practical situations. Furthermore, 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.

2. Objective

The goal of this work was to develop a modified version of the SDC coder that 1) removes any restrictions on the image width/height and sample-precision; 2) offers improved coding efficiency; and 3) most importantly provides an efficient progressive lossy-to-lossless coding capability.

3. Arbitrarily-Sampled Image Dataset

Consider a grayscale image defined on a rectangular domain $\mathcal{D}$ of width W and height H (i.e., $\mathcal{D} = [0, W) \times [0, H)$) and having integer-valued samples in the range $[0, D)$. For such an image, an **arbitrarily-sampled dataset** is a set of **sample positions** $\{p_i = (x_i, y_i)\}$ and their corresponding **sample values** $\{z_i\}$, where $p_i \in \mathcal{D}$, $z_i \in [0, D)$, and x_i and y_i denote the horizontal and vertical position of p_i, respectively. The sample positions $\{p_i\}$ must be distinct.

4. SDC Coder

The SDC coder [1] views an arbitrarily-sampled image 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 $\mathcal{I}$. The coding scheme employed by the SDC coder is based on an octree partitioning of $\mathcal{I}$ into hyperrectangular regions called cells. As a matter of terminology, the **volume** of a cell is defined as the number of lattice points (from $\mathbb{Z}^3$) it contains. As 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. A cell with a volume of four or less is called **atomic**.

To begin encoding, a header is written with W, H, and D, and the total number of sample points. Then, the root cell $\mathcal{I}$ is recursively split to produce the remainder of the code stream. The cell splitting process works as shown in Fig. 1. A cell C is split first in the x direction yielding two cells (i.e., $\{C'_i\}_{i=0}^{1}$), then in the y direction yielding four cells (i.e., $\{C''_i\}_{i=0}^{3}$), and finally in the z direction yielding eight cells (i.e., $\{C_i\}_{i=0}^{7}$). As each of the cells $C, C'_0, C'_1, C''_0, C''_1, C''_2, C''_3$ is split, the number of sample points contained in one of the two resulting cells is coded. For each child cell Q in $\{C_i\}_{i=0}^{7}$, if Q is empty or full, we do nothing; otherwise, we proceed as follows. If Q is atomic, the configuration of the sample points within Q is directly coded. Otherwise, Q is recursively split.

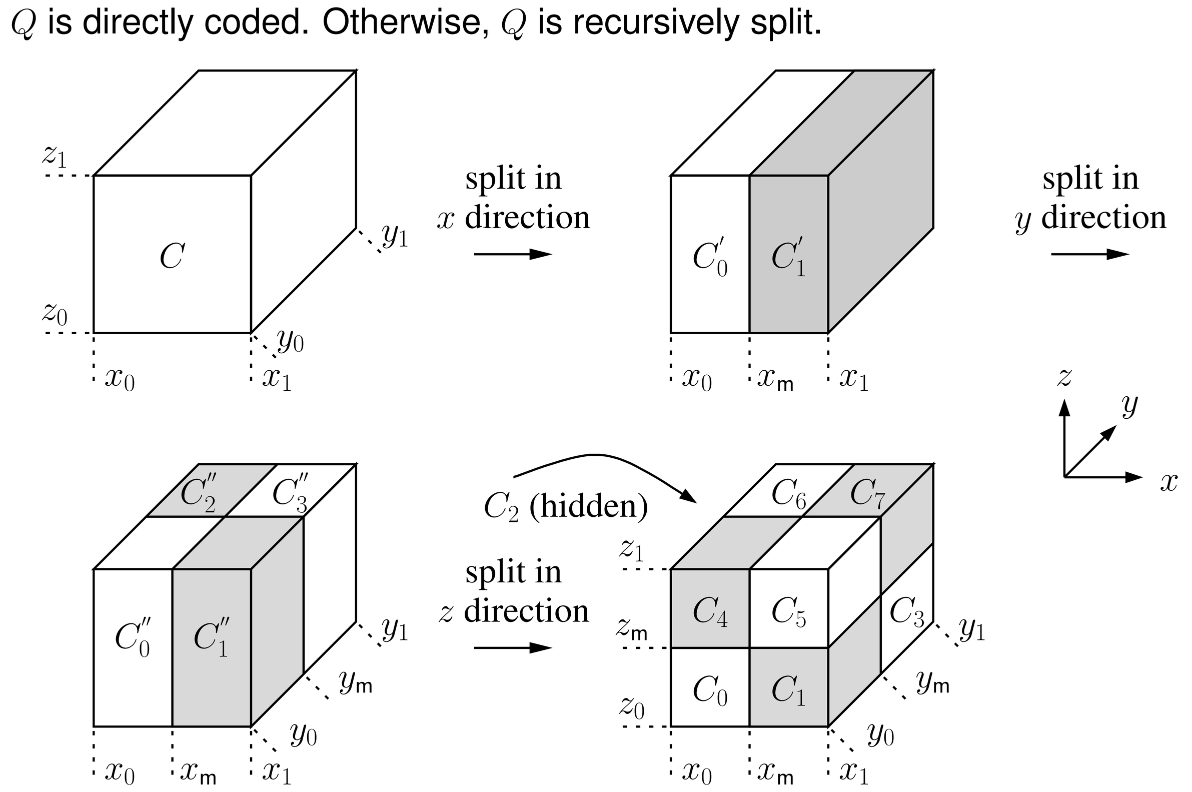

Fig. 1: *Cell splitting. A cell C is split in the x, y, and z directions in order to produce eight child cells $\{C_i\}_{i=0}^{7}$.*

5. Modified SDC (MSDC) Coder

In our work, we have proposed an improved version of the SDC coder called the modified SDC (**MSDC**) coder. The **key differences** between our MSDC coder and the SDC coder can be summarized as follows:

1. To allow for arbitrary image width/height and sample precision in the MSDC coder, the formula for the cell midpoint used in cell splitting has been changed to include a rounding operation.
2. To provide a flexible progressive-coding capability in the MSDC coder, the cells still remaining to be processed are kept in a priority queue, called the **work queue**, and cells are processed in the order that they are removed from this queue (i.e., the highest priority cells are processed first). By using different cell-priority functions, the order in which information is coded can be controlled, leading to different progression orders.
3. The MSDC coder employs arithmetic coding, while the SDC coder employs Huffman coding.

As a consequence of item 1 above, the MSDC coder has a number of other differences with the SDC coder. For example, in the MSDC coder, the cell splitting process can result in degenerate cells, whereas the SDC coder never encounters such cells. Also, in the MSDC coder, there are more possible sizes for atomic cells. These differences result in numerous new cases in the MSDC coder that must be identified and handled correctly.

Progressive Coding. During the coding process, cells are split into smaller cells until the locations of all of the sample points are known exactly. As soon as all of the sample-point locations are known exactly to the decoder, it can losslessly reconstruct the dataset. The decoder, however, can obtain approximations to the dataset before all of the sample-point locations are known exactly. If the locations of the sample points in a particular cell are not known exactly, the decoder can simply choose to represent the sample points in the cell with a single **representative sample point** at the approximate centroid of the cell. In this way, progressive coding functionality can be provided. Furthermore, different progression orders can be achieved by varying the cell-priority function used with the work queue.

Sample-Value Ambiguity Problem. Unfortunately, there is a fundamental problem associated with progressive decoding that must be addressed in order to obtain good coding performance. Namely, it is possible, during intermediate stages of decoding, for two or more nonempty cells to have representative sample points with the same x and same y coordinates but *distinct* z coordinates. This would correspond to a particular sample position having multiple distinct sample values, which is clearly impossible. In such a situation, the decoder must resolve this ambiguity and choose a single z value (i.e., sample value) for the sample position of interest. This leads to the question of how best to resolve this ambiguity problem. In our work, we have proposed the following four sample-value ambiguity-resolution methods:

1. **Discard**. Throws away any ambiguous sample points.
2. **Mean**. Uses the mean of the conflicting sample values.
3. **Nearest neighbour**. Chooses the sample value that deviates least from the sample value of the closest neighbouring ambiguity-free sample point.
4. **Median**. Uses the median of the conflicting sample values.

Progression Order. Since many progression orders are possible, one might wonder which yields the best coding performance. In our work, we have considered the following six progression orders:

1. **Breadth first**. The cell that is the nearest descendant of the root cell is split first.
2. **Depth first**. The cell that is the farthest descendant of the root cell is split first.
3. **Count**. The cell containing the most sample points is split first.
4. **Density**. The cell with the highest sample-point density is split first.
5. **Sparsity**. The cell with the lowest sample-point density is split first.
6. **Deviation from half density (DFHD)**. The cell with the sample-point density that deviates most from $1/2$ is split first.

With each of these progression orders, the next information to be decoded can always be determined from previously decoded data. Therefore, the progression order itself does not need to be coded as side information.

6. Experimental Results

Experimentally, we studied how the choices of sample-value ambiguity-resolution method and progression order affect our coder, and evaluated our coder's performance relative to the SDC coder. Here, we present a representative subset of our results. In our experiments, we used the MGH mesh-generation method and corresponding triangulation-based interpolation scheme from [2] to generate arbitrarily-sampled datasets from (lattice-sampled) images and vice versa.

Sample-Value Ambiguity Resolution. Progressive coding results comparing the performance of the four proposed sample-value ambiguity-resolution methods are given in Fig. 2, with one set of lossy image reconstructions shown in Fig. 4. From the results of Fig. 2, we can see that the median method performs best, often beating the other methods by more than 1 dB. As illustrated by Fig. 4, in terms of subjective image quality, the median method also performs best. Due to the sample-value ambiguity problem, the rate-distortion curves for our MSDC coder can depart significantly from monotonic behavior, especially at low rates. This behavior is clearly evident in the graph (i.e., Fig. 2).

Progression Order. Progressive coding results comparing the effectiveness of the six progression orders under consideration are given in Fig. 3, with one set of lossy image reconstructions shown in Fig. 5. From the results of Fig. 3, we can see that the sparsity and DFHD progressions orders (which yield almost identical rate-distortion curves) outperform the other progression orders by a significant margin (i.e., 0.5 to 1 dB or more). As is evident from the reconstructed images shown in Fig. 5, the DFHD progression order also yields the best subjective image quality. Furthermore, a more detailed evaluation shows that the DFHD progression order offers slightly better objective performance at very high rates.

MSDC Coder Versus SDC Coder. Results comparing the lossless coding performance of our MSDC coder and the SDC coder are provided in Table 1. From these results, we can observe that our MSDC coder yields a lossless rate that is typically about 4% less than that obtained with the SDC coder.

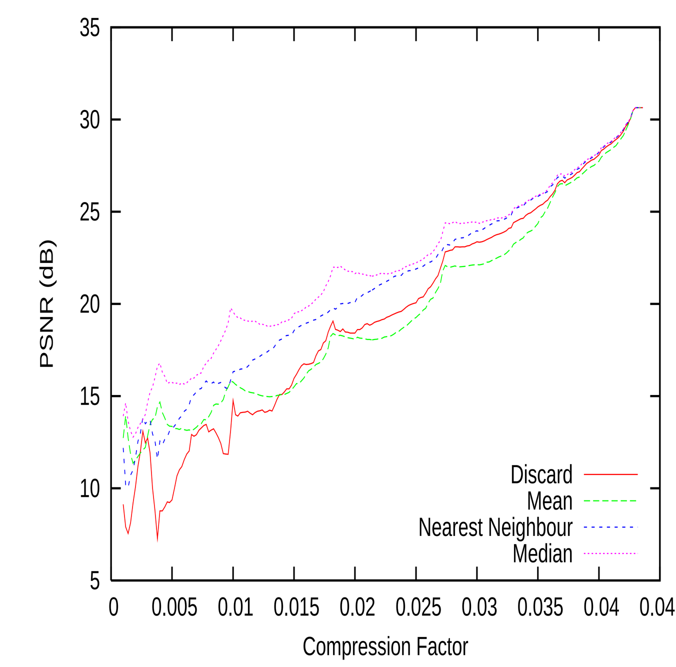

Fig. 2: **Comparison of various sample-value ambiguity-resolution methods.** *Progressive coding results obtained using various sample-value ambiguity-resolution methods for the `lena` image with a dataset sampling density of 1/40.*

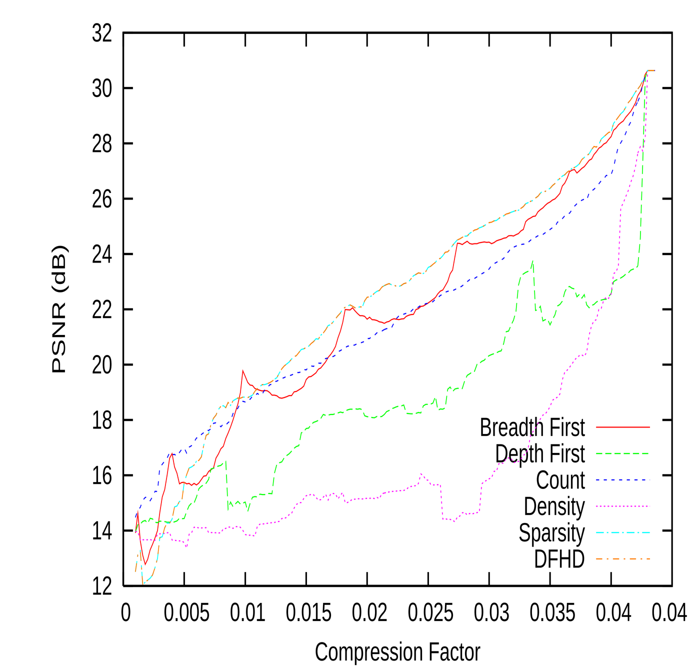

Fig. 3: **Comparison of various progression orders.** *Progressive coding results obtained using various progression orders for the `lena` image with a dataset sampling density of 1/40.*

(a) (b) (c) (d)

Fig. 4: **Subjective image quality comparison for various sample-value ambiguity-resolution methods.** *Lossy reconstructions obtained at about 33:1 compression with the (a) discard (23.34 dB), (b) mean (22.11 dB), (c) nearest-neighbour (23.92 dB), and (d) median (24.37 dB) sample-value ambiguity-resolution methods.*

(a) (b) (c) (d) (e) (f)

Fig. 5: **Subjective image quality comparison for various progression orders.** *Lossy reconstructions obtained at about 29:1 compression with the (a) breadth-first (25.88 dB), (b) depth-first (21.43 dB), (c) count (24.88 dB), (d) density (18.48 dB), (e) sparsity (26.37 dB), and (f) DFHD (26.37 dB) progression orders.*

Table 1: **Comparison of MSDC and SDC coders.** *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

7. Conclusions

We have proposed the MSDC coder, a modified version of the SDC coder. Relative to the SDC coder, our MSDC coder has three **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.

Acknowledgments

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

References

[1] 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.

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

[3] 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.