Successive Approximation Wavelet Coding of AVIRIS Hyperspectral Images

CIPR Technical Report TR-2010-3

Alessandro J. S. Dutra, William A. Pearlman, and Eduardo A. B. da Silva

August 2010

Center for
Image Processing Research

Rensselaer Polytechnic Institute
Troy, New York 12180-3590
http://www.cipr.rpi.edu

Successive Approximation Wavelet Coding of AVIRIS Hyperspectral Images

Alessandro J. S. Dutra, *Member, IEEE,* William A. Pearlman, *Fellow, IEEE,* and Eduardo A. B. da Silva, *Senior Member, IEEE*

Abstract

This work presents lossy compression algorithms which build on a state-of-the-art codec, the Set Partitioned Embedded Block Coder (*SPECK*), by incorporating a lattice vector quantizer codebook, therefore allowing it to process multiple samples at one time. In our tests, we employ scenes derived from standard AVIRIS hyperspectral images, which possess 224 spectral bands.

The first proposed method, LVQ-SPECK, uses a lattice vector quantizer-based codebook in the spectral direction to encode a number of consecutive bands that is equal to the codeword dimension. It is shown that the choice of orientation codebook used in the encoding greatly influences the performance results. In fact, even though the method does not make use of a 3D discrete wavelet transform, in some cases it produces results that are comparable to those of other state-of-the-art 3D codecs.

The second proposed algorithm, DWP-SPECK, incorporates the 1D discrete wavelet transform in the spectral direction, producing a discrete wavelet packet decomposition, and simultaneously encodes a larger number of spectral bands. This method yields performance results that are comparable or superior to those attained by other 3D wavelet coding algorithms such as 3D-SPECK and JPEG2000 (in its multi-component version).

We also look into a novel method for reducing the number of codewords used during the refinement pass in the proposed methods which, for most codebooks, provides a reduction in rate while following the same encoding path of the original methods, thereby improving their performance.

A. J. S. Dutra is with the PEE–COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil. (corresponding author)

W. A. Pearlman is with the ECSE Department, Rensselaer Polytechnic Institute, Troy, NY. His work was supported in part by the Office of Naval Research under Grant No. N00014-05-1-0507

E. A. B. da Silva is with the PEE–COPPE/DEL–Poli, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil.

We show that it is possible to separate the original codebook used into two distinct classes, and use a flag when sending refinement information to indicate to which class this information belongs.

In summary, given the results obtained by our proposed methods, we show that they constitute a viable option for the compression of volumetric datasets with large amounts of data.

I. Introduction

Several data compression algorithms have been proposed in the last few years as the demands for digital storage and transmission of information have become increasingly large. In this article, we examine how the use of Lattice Vector Quantization (LVQ) affects the performance behavior of quadtree-based methods for lossy coding. The subject is brought up by the vast number of data sources that can be categorized as vector-valued ones including, but not limited to, volumetric medical images, hyperspectral images, video sequences etc.

Although very good performance can be obtained with the use of current state-of-the-art scalar codecs [1]–[3], it remains true that jointly encoding a group of samples, i.e. vector quantizing them, should provide better results, even though at a complexity cost.

Lattice vector quantization has been successfully used in the compression of both still-images [4], [5] and video sequences [6]. By employing LVQ, we hope to achieve a balance between performance and complexity, knowing that not all coding gain improvements will be achieved, but keeping complexity down by avoiding the training phases that standard vector quantizers have to go through.

The advent of the *discrete wavelet transform* (DWT) [7], [8] and subband-based decomposition systems in general [9], and their recognition as potential substitutes for the then ubiquitous *discrete cosine transform* (DCT) [10] led to the development of several compression systems.

Amongst wavelet coders, the *Embedded Zerotree Wavelet* (EZW) codec [11] was the first one to achieve very good results, while managing to keep complexity at a very low level. It employed a tree structure to bundle groups of samples from across the decomposed subbands (*interband* encoding), representing a given spatial location on the original image that, at the current encoding threshold level, would be deemed insignificant and, therefore, coded as zero. Hence the name *zerotree*.

An improvement over EZW was obtained by the *Set Partitioning in Hierarchical Trees* (SPIHT) codec [2], which set up spatial orientation trees of coefficients in a more efficient way, resulting in a better exploitation, and consequently removal, of intrinsic redundancies present in the data to be

coded. This fact was evidenced by the very small gains obtained when further entropy encoding SPIHT's output with an arithmetic coder [12].

Switching from an interband to an *intraband* setting, the *Set Partitioning Embedded Block Coder* (SPECK) [1], [13] looked to rapidly converge on significant pixels (wavelet coefficients), while grouping non-significant adjacent samples within a subband into coding blocks.

The list of most relevant state-of-the-art still image compression systems is completed by the *Embedded Block Coder with Optimized Truncation* (EBCOT) codec [3], which is the main encoding algorithm behind the JPEG2000 image compression standard [14].

All of the above codecs were initially designed to process two-dimensional images. However, given the numerous applications that involve the use of three- or higher dimensional datasets, it was only a matter of time before extensions for higher dimensional versions of all of them were proposed. Amongst them we may cite the 3D-SPIHT [15] and 3D-SPECK [16] codecs and the *Multi-Component Imaging* option of the JPEG2000 standard [14] (JPEG2000 Part 2).

A. Outline

In Section II we present a brief definition of hyperspectral images along with a description of the AVIRIS [17] dataset and review some of the previous work regarding compression of those datasets. We also define the figure of merit that will be used throughout the article whenever a quantitative comparison is presented.

Section III introduces the concept of lattices and discusses some of their properties, especially those important for their use as quantizers. We then proceed to review a few compression algorithms, including codecs which have served as basis for our development.

The compression of higher dimensional datasets is the subject of Section IV. After a brief review of the partitioning routines of the SPECK algorithm, we introduce the LVQ-SPECK algorithm, which extends the concepts of block set-partitioning present in SPECK to simultaneously encode a number of consecutive spectral bands in a hyperspectral image volume. This is accomplished with the use of a lattice vector quantizer applied in the spectral direction.

A second algorithm, DWP-SPECK, adds a 1D discrete wavelet transform in the spectral direction to produce a discrete wavelet packet decomposed dataset. By using this decomposition and processing yet a larger number of consecutive bands, the method attains very good compression performance.

In Section V, we introduce a new technique to reduce the number of codewords used in the

Image Name	Image Type	Power (P_x)
Cuprite (Scene 01)	Radiance	6306786
Cuprite (Scene 04)		7141669
Jasper Ridge (Scene 01)		2583295
Jasper Ridge (Scene 03)		2458457
Moffet Field (Scene 03)	Reflectance	2319634

TABLE I

Characteristics of AVIRIS hyperspectral image volumes (512×512×224 pixels, 16-bit integers)

refinement codebook. In particular, we separate the original codebook into two distinct classes which are selected according to whether the best reproduction codeword points towards the outside or the inside of the current encoding hypersphere. Results show that for most choices of codebook, there is an improvement in the performance of both LVQ-SPECK and DWP-SPECK.

Lastly, Section VI presents a summary of the work herein developed.

II. Review and Definitions

A. Hyperspectral Images

A hyperspectral image is a dataset which contains a given scene observed through a large number (usually, in the hundreds) of wavelengths. Therefore, such a remote sensing operation produces, for each pixel of the scene, its spectrum.

In the case of AVIRIS hyperspectral images, each run of the airborne sensors produces scenes which have 224 spectral bands. The length of a run is not defined *a priori* but, to keep storage of the raw data manageable, each strip is divided into 512 pixels long *scenes*. For each band, the value of each pixel is stored as a 16-bit integer.

Table I presents characteristics of the datasets that will be used throughout this work. It should be noted that each scene has been further cropped to a 512×512×224 block, so that comparison with other existing methods can be more easily made.

B. Existing Compression Methods

A lot of attention has been given to compressing hyperspectral images, due not only to the often sensitive nature of the acquired information but also because of the usually large amount of data

needed to represent it, producing a vast amount of literature on the subject, e.g. [18], [19], [20] and [21]. A thorough review of 3D wavelet-based encoders is presented in [22].

Methods spanning from direct quantization of spectral values [23], [24] to those that employ the discrete wavelet transform [25] as a decorrelating step were developed, providing good compression capabilities along with good quality representation. Several methods also provide lossless compression capabilities.

In [23], [24], Motta et al. define a partition of the spectral space whose boundaries are optimized by repeated application of a Generalized Lloyd Algorithm [26] (GLA) variant. Considering the original data set to have a dimension D, the design of a D-dimensional vector quantizer, which is usually computationally prohibitive, would be required. Instead, the method chooses to design N vector quantizers, each with dimension d_i, where $\sum_{i=1}^{N} d_i = D$.

The resulting Partitioned Vector Quantizer is then the Cartesian product of all the lower dimensional dictionaries. In order to remove part of the remaining source redundancy, each resulting vector quantization (VQ) index is also conditionally entropy encoded based on a causal set of spatially and spectrally adjacent indices.

As opposed to the previously described method, which encodes the spectral band intensity values directly, a number of methods that apply a decorrelating transform were developed. In [25], a 3D version of the quadtree-based codec SPECK [1] was introduced.

3D-SPECK divides the hyperspectral block into sub-blocks of 16 spectral bands at a time, applies a 3D discrete wavelet transform (DWT) and extends the concepts of partitioning sets and rules to the three-dimensional case. Given the energy compaction properties of the DWT, and SPECK's efficiency in the coding of significance information, the method achieves very good compression results.

A multidimensional version of SPIHT that employs the multistage lattice vector quantization (MLVQ) encoding concept described in [27] was also used to compress hyperspectral images. In that work, different vector quantizers are defined by varying both the four-dimensional lattice and the significance measure being used. Three different options are explored – the cubic lattice $\mathbf{Z}^4$ coupled with the L_∞ norm and the D_4 lattice with both the L_1 and L_2 norms.

Perhaps among the best results presented so far in compressing AVIRIS [17] datasets are those of algorithms that employ PCA decomposition in the spectral direction as a pre-quantization transform [28]. However, the use of an optimal transform such as the KLT implies in a preprocessing stage prior to encoding every dataset and expanded buffering capabilities, as the whole

dataset must be processed at once.

C. Quantizer Performance

Even though its relation to any subjective quality assessment has been regarded as questionable in many areas [29], [30], when analyzing the encoding performance of the proposed methods we will use the signal-to-noise ratio (SNR) function, defined as

$$\mathrm{SNR}(\boldsymbol{x}, \hat{\boldsymbol{x}}) \quad = \quad 10 \log \frac{P_x}{d_{\mathrm{MSE}}(\boldsymbol{x}, \hat{\boldsymbol{x}})} \tag{1}$$

where P_x is the mean square value of the input signal $\mathcal{X}$ and $d_{\mathrm{MSE}}(\boldsymbol{x}, \hat{\boldsymbol{x}})$ is the mean square error (MSE) function

$$d_{\mathrm{MSE}}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = E[(\boldsymbol{x} - \hat{\boldsymbol{x}})^2]. \tag{2}$$

It should be noted that although many in the literature use the signal variance instead of its mean square value, it has been pointed out in [19] that, in the case of nonzero mean valued data, such as hyperspectral images, coupled with the use of (close to) energy-preserving transforms during the coding/decoding phases, it makes sense to define the performance metric as in Eq. 1, thereby accounting for the effect of the DC component.

III. Successive Approximation of Vectors

A. Lattices and Their Characteristics

A lattice [31] Λ is a set of points $\boldsymbol{x}$ in the n-dimensional space $\mathbb{R}^n$ such that

$$\boldsymbol{x} = a_1 \boldsymbol{v}_1 + a_2 \boldsymbol{v}_2 + \ldots + a_n \boldsymbol{v}_n, \tag{3}$$

where $\{\boldsymbol{v}_i\}$ is a basis for $\mathbb{R}^n$ and $a_i \in \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers and $\mathbb{R}$ is the set of real numbers. Fig. 1 shows an example of a lattice on the Cartesian plane.

Given a lattice Λ, we define a lattice shell around a given point $\boldsymbol{x}_0$ as the set

$$\Lambda_{\mathrm{shell}}(\boldsymbol{x}_0) = \{\boldsymbol{x} \in \Lambda : \|\boldsymbol{x} - \boldsymbol{x}_0\| = l, l \in \mathbb{R}\}, \tag{4}$$

that is, the set of lattice points that are equidistant to a given *origin* $\boldsymbol{x}_0$. Without any loss of generality, we will take $\boldsymbol{x}_0 = \overrightarrow{0}$, the coordinate system's origin.

It is known that the distribution of coefficients of a DWT-transformed image is usually considered to have a Generalized Laplacian shape, which would be better approximated with the use of the L_1-norm if direct quantization of the coefficients were used. However, during the successive

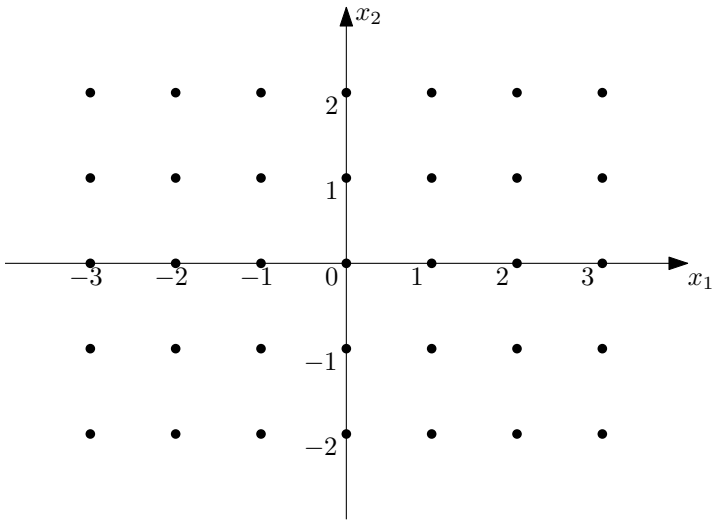


Fig. 1. $\mathbf{Z}^2$ lattice

approximation process used in the proposed algorithms, the distribution of residues (which are the information actually being encoded) cannot be considered Laplacian, presenting a more uniformly distributed shape [32], [33] and, therefore, the L_2-norm (Euclidian) used in the shell definition presents a more suitable alternative.

For a shell, the maximum angular distance between any given point and the closest available shell point is denoted by $\theta_{\max} = \Theta_{\max}(\Lambda_{\text{shell}})$ and defined as [34]

$$\Theta_{\max}(\Lambda_{\text{shell}}) = \cos^{-1}\left\{\max_{\boldsymbol{x}\in\mathbb{R}^n}\left\{\min_{\boldsymbol{u}_i\in\Lambda_{\text{shell}}}\left(\frac{\langle \boldsymbol{x}, \boldsymbol{u}_i\rangle}{\|\boldsymbol{x}\|\,\|\boldsymbol{u}_i\|}\right)\right\}\right\}, \tag{5}$$

where $\langle \boldsymbol{x}, \boldsymbol{u}_i\rangle$ denotes the inner product of the vectors $\boldsymbol{x}$ and $\boldsymbol{u}_i$.

Therefore, if a given shell Λ_{shell} is used as a quantizer, an n-dimensional source input vector will make, at most, an angle of $\theta_{\max}$ with the closest quantizer codevector. For the methods herein proposed, as will be seen in Sec. III-B, for a given dimension n the lattice with the smallest $\theta_{\max}$ is the one that presents the best covering properties and will be selected as the orientation codebook.

It has also been shown in [34] that for every Λ_{shell} that is used as a quantizer, and its corresponding $\theta_{\max}$, there exists a quantity, which shall be denoted α_{opt}, that defines the fastest contraction rate to which Λ_{shell} may be submitted to, such that the union of the Voronoi regions of the original shell and those of subsequent contractions will cover the entire n-dimensional space.

Even though lattices are available for any number of dimensions, we will be interested in only a few ones, whose characteristics make them more appropriate for use in a successive approximation

setting, and also in the sense that it provides us with results which are more suitable to be compared against those already presented in the literature. Particularly, in our experiments we will use two different shells of the D_4 lattice and the first shell of the E_8 and Λ_{16} lattices. A short description follows:

- $\mathbf{Z}^n$ lattice: The n-dimensional cubic (or integer) lattice, defined as

$$\mathbf{Z}^n = \{(x_1, \ldots, x_n) : x_i \in \mathbb{Z}\}, \tag{6}$$

  will be used as the basic building block in the definitions that follow. Being an n-dimensional Cartesian product of the set of integers, it is the simplest regular arrangement defined.

- D_n lattice: Lattices of type D_n are extracted from integer lattices $\mathbf{Z}^n$ by keeping only those points whose sum of coordinates is a multiple of 2. Formally, we have:

$$D_n = \{(x_1, \ldots, x_n) \in \mathbb{Z}^n : x_1 + \ldots + x_n \equiv 0 \pmod 2\}, \tag{7}$$

  In four dimensions, we will be particularly interested in the 1^{st} and 2^{nd} shells of the D_4 lattice. The 1^{st} shell contains vectors of the form $(\pm 1, \pm 1, 0, 0)$ (or equivalently $(\pm 1^2, 0^2)$) and its permutations, which have a $\sqrt{2}$ norm. The 2^{nd} shell, on the other hand, presents two different types of vectors, both of which have norm 2, namely $(\pm 2, 0, 0, 0)$ and $(\pm 1, \pm 1, \pm 1, \pm 1)$. Both shells are comprised of 24 vectors.

- E_n lattice: E_n lattices provide the densest regular arrangement of points for dimensions 6, 7 and 8. For $n = 8$, the Gosset lattice E_8 is defined as

$$\begin{aligned} E_8 = \{(x_1, \ldots, x_8) \quad : \quad & \text{all } x_i \in \mathbb{Z} \text{ or all } x_i \in \{\mathbb{Z} + \tfrac{1}{2}\}, \\ & \sum x_i \equiv 0 \pmod 2\}. \end{aligned} \tag{8}$$

  The first shell of the E_8 lattice has 240 vectors of norm 2, being comprised of vectors of both the $(\pm 1^2, 0^6)$ and $(\pm \frac{1}{2}^8)$ forms.

- L_n lattice: For $n = 16$ we will be mostly interested in the Λ_{16} Barnes-Wall lattice, which is the densest at this dimension. Λ_{16} is defined in terms of unions of 32 cosets of the scaled $2D_{16}$ lattice. Formally,

$$\Lambda_{16} = \bigcup_{i=1}^{32} \{2D_{16} + \boldsymbol{c}_i\} \tag{9}$$

  where the vectors $\boldsymbol{c}_i$ are the rows of the Hadamard matrix $\mathbf{H}_{16}$ and its complementary $\overline{\mathbf{H}}_{16}$, defined as in Eq. 10.

$$\left[\begin{array}{c}\mathbf{H}_{16}\\ \hline \overline{\mathbf{H}}_{16}\end{array}\right] = \left[\begin{array}{cccccccccccccccc}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\
1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1\\
1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0\\
1&0&1&0&0&1&0&1&1&0&1&0&0&1&0&1\\
1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1\\
1&0&0&1&0&1&1&0&1&0&0&1&0&1&1&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&1&0&1&0&1&0&1\\
1&1&0&0&1&1&0&0&0&0&1&1&0&0&1&1\\
1&0&0&1&1&0&0&1&0&1&1&0&0&1&1&0\\
1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1\\
1&0&1&0&0&1&0&1&0&1&0&1&1&0&1&0\\
1&1&0&0&0&0&1&1&0&0&1&1&1&1&0&0\\
1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1\\
\hline
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1\\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\
0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0\\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\
0&1&0&1&1&0&1&0&0&1&0&1&1&0&1&0\\
0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0\\
0&1&1&0&1&0&0&1&0&1&1&0&1&0&0&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&1&0&1&0&1&0&1&1&0&1&0&1&0&1&0\\
0&0&1&1&0&0&1&1&1&1&0&0&1&1&0&0\\
0&1&1&0&0&1&1&0&1&0&0&1&1&0&0&1\\
0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0\\
0&1&0&1&1&0&1&0&1&0&1&0&0&1&0&1\\
0&0&1&1&1&1&0&0&1&1&0&0&0&0&1&1\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0
\end{array}\right] \tag{10}$$

A brief summary of their properties is presented on Table II.

Lattice	Shell	Cardinality	$\theta_{\max}$
D_4	1	24	45^o
D_4	2	24	45^o
E_8	1	240	45^o
Λ_{16}	1	4320	55^o

TABLE II

LATTICE PROPERTIES SUMMARY

B. Vector Extensions for Wavelet Codecs

In extending a scalar codec to deal with vector-valued samples, there usually is no reason to discard certain classes of vector quantizers from being used as codebooks. In practice, however, complexity issues associated with the use of unconstrained (e.g. LBG-optimized [35], [36]) vector quantizers, especially at higher dimensions, lead to the use of structured types of codebooks, such as tree-structured and lattice-based ones.

Amongst those, we will be particularly interested in lattice vector quantizers, given that they possess characteristics that are desired in a set of n-dimensional points that is to be used as a codebook in a successive approximation type codec [37], such as:

1. for a given vector dimension n, the value of $\theta_{\max}$ for the chosen shell-based lattice vector quantizer should be smaller than those of codebooks based on other lattices and, in particular, smaller than that of the $\mathbf{Z}^n$ lattice's 1^{st} shell.
2. considering different shells of the same lattice, the higher the cardinality of a shell, the smaller the value of $\theta_{\max}$ and, consequently, the number of refinement passes needed until convergence.
3. the use of codewords of larger dimension generates codebooks of higher cardinality and value of $\theta_{\max}$. However, that is offset by a decrease in the number of bits per dimension needed.

Based on the preceding argument, it is expected that constant-norm shells (or combination thereof) of regular lattices should be considered as natural candidates in the definition of codebooks – which will also be denoted *orientation* codebooks – for use in successive approximation methods.

We now review two vector-based still-image codecs whose codebooks are based on the lattices herein discussed.

1) The SAWVQ Approach – Generalized Bitplanes: The Successive Approximation Wavelet Vector Quantizer (SAWVQ) [38] algorithm is a vector-based extension of the EZW codec [11] –

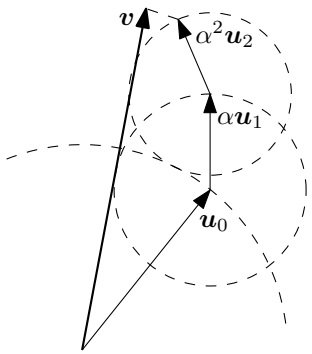


Fig. 2. Successive approximation of vectors using an orientation codebook.

and therefore also mentioned in the literature as VEZW – that employs a shell-based, orientation codebook as its quantizer.

Formally, a vector $\boldsymbol{v}$, $\|\boldsymbol{v}\| \leq 1$, is said to be successively approximated by a sequence of codewords $\boldsymbol{u}_l$ if the summation

$$\boldsymbol{v} = \sum_{l=0}^{\infty} \alpha^l \boldsymbol{u}_l, \tag{11}$$
$$\boldsymbol{u}_l \in C = \{\boldsymbol{c}_0, \boldsymbol{c}_1, ..., \boldsymbol{c}_K\}$$

converges, where C is the codebook, $\boldsymbol{c}_k$ are the codewords and α $(0 < \alpha < 1)$ is a scaling factor to account for the fact that after each interaction the residual error is bound by a smaller N-dimensional hypersphere.

As mentioned in Sec. III-A, for every codebook – and codeword dimension – there is a choice (often empiric) of α that proves to be optimal, i.e., that provides the best representation results.

Since in lossy coding we are interested only in obtaining a close enough approximation of the original data, that is, with a limited amount of error, a finite summation is used instead of the infinite one, resulting in

$$\boldsymbol{v}_L = \sum_{l=0}^{L} \alpha^l \boldsymbol{u}_l. \tag{12}$$

Given that this vector approximation method is a direct extension of the scalar bitplane encoding concepts used by SPIHT [2] and other scalar codecs, we shall henceforth denote it *generalized bitplane encoding* of vectors. The process is depicted in Fig. 2.

It was also shown in [34] that this type of method for successively approximating vectors is guaranteed to converge in a finite amount of time.

For more information on the SAWVQ algorithm, the reader is referred to [38].

2) The VSPIHT Approach – VLVQ Encoding: Mukherjee and Mitra [39], [40] also presented a structured method for successive refinement of vectors, in which scaled versions of a given lattice are used as quantizers over each step of the approximation process. However, unlike SAWVQ, VSPIHT employs a different codebook populating strategy.

Instead of choosing constant-norm sets of vectors (shells) from a given multidimensional lattice, the VSPIHT algorithm truncates the lattice to a given number of points around the origin and uses those as codevectors.

The encoding process in VSPIHT is based on Voronoi region approximation and the codebook is based on the following definitions:

- Base lattice (Λ_1): lattice coset from which the codebook is actually derived.
- Shape lattice (Λ_0): higher scale lattice which determines the shape of the codebook.

The resulting quantizer, called *Voronoi Lattice Vector Quantizer*, is therefore defined as

$$\mathrm{VLVQ}(\Lambda_0, \Lambda_1) = V_0(\Lambda_0) \cap \Lambda_1 \tag{13}$$

where $V_0(\Lambda_0)$ is the zero-centered Voronoi region associated with the lattice. The shape lattice is defined so that it covers the n-dimensional region of support of the data source and, in the most common case, the base lattice is just a scaled down and (possibly) translated version of the shape lattice, i.e.,

$$\Lambda_1 = \frac{\Lambda_0}{r} - \boldsymbol{t}, \tag{14}$$

$\boldsymbol{t}$ being the translation vector.

More detailed descriptions of the VSPIHT algorithm can be found in [39], [40].

IV. Wavelet Coding of Hyperspectral Images

A. The SPECK Partitioning Routines

Instead of exploiting the interband similarities of wavelet coefficients as done by SPIHT [2] and other codecs, the Set Partitioned Embedded Block Coder (SPECK) [1], [13] employs a different approach, defining sets based on blocks of coefficients from within a single subband of the transform data, and defining trees in terms of recurrent splitting of these blocks.

Considering that the approximation process in SPECK, i.e. both the sorting and the refinement steps, are spatially restricted to a given subband, the algorithm inherently possess a resolution scalability feature. In fact, a given $n \times n$ image may be reconstructed to size $n/2 \times n/2$ simply by ignoring the highest frequency subbands.

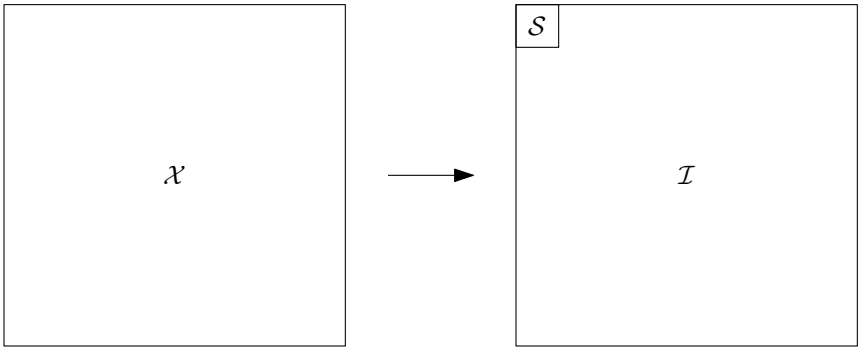


Fig. 3. Initial data partition – SPECK algorithm

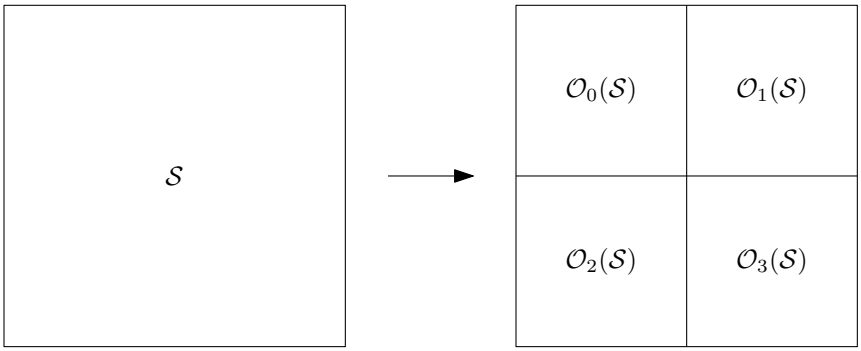


Fig. 4. $\mathcal{S}$-type set partitioning in SPECK

The two types of sets used by SPECK are referred to as $\mathcal{S}$ and $\mathcal{I}$ sets. $\mathcal{S}$ sets are rectangular blocks of the image (hence the name *Block Coder*), of varying dimensions, that depend on the size of the original image and the level of the pyramid decomposition to which the set belongs. Fig. 3 shows the initial assignment of $\mathcal{S}$ and $\mathcal{I}$ sets. After decomposing the input image in subbands with the application of a DWT, the set $\mathcal{X}$, which has the same dimensions of the original data, is divided in two sections. The initial set $\mathcal{S}$ is assigned to the region corresponding to the LL frequency subband with coarsest scale. To the remaining piece of data, corresponding to $\mathcal{X} - \mathcal{S}$, is assigned the initial set $\mathcal{I}$.

SPECK also uses lists – managed identically by both the encoder and the decoder – to classify sets and implement the quantization step as well. SPECK maintains only two lists, the List of Insignificant Sets (LIS) and the List of Significant Pixels (LSP).

After submitting the input data to the subband decomposition, encoding proceeds by setting up the initial $\mathcal{S}$ and $\mathcal{I}$ sets, as previously mentioned. In the sorting pass, each currently existing set

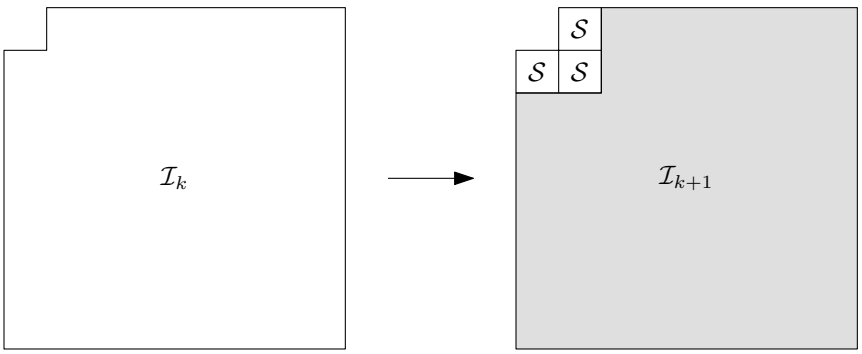


Fig. 5. $\mathcal{I}$-type set partitioning in SPECK

of type $\mathcal{S}$ is processed by the function `ProcessS()`, which checks the set for significance against the current encoding threshold. If the set is declared insignificant, it is moved to the LIS.

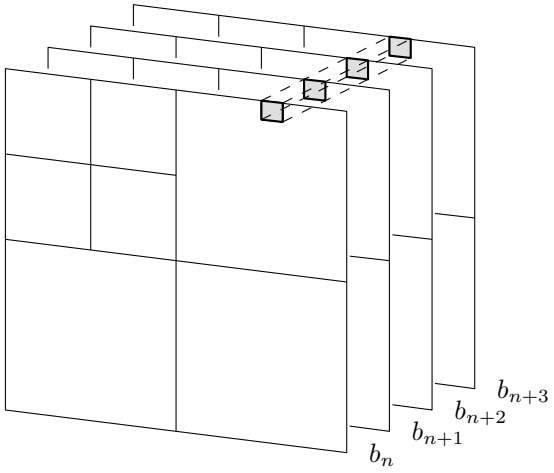
However, when a set is found to be significant, two options are available. The first one treats those sets that are made of a single pixel, by outputting the sign of that coefficient and moving it to the LSP. If the set is comprised of a group of pixels, then the function `CodeS()` (to be described soon) is called to further process it. In order to correctly maintain the lists, if the set was previously in the LIS, it is then removed from it.

The `CodeS(`$\mathcal{S}$`)` function takes a type $\mathcal{S}$ set as its input and divides it into four offspring $\mathcal{S}_i = \mathcal{O}_i(\mathcal{S}), i = 0, 1, 2, 3$, that are also sets of type $\mathcal{S}$, but with half the linear dimensions of the parent set, as shown in Fig. 4. `CodeS()` also outputs the results of the significance test for each $\mathcal{S}_i$, and adds them to the LIS whenever applicable. If the $\mathcal{S}_i$ is deemed significant, again there are two possibilities: if $\mathcal{S}_i$ is a pixel, its sign is added to the bitstream and the set is added to the LSP. However, if it is not a pixel, another recursion of the `CodeS(`$\mathcal{S}_i$`)` is called.

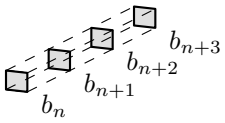
The two remaining functions used by SPECK involve processing the $\mathcal{I}$ set. The `ProcessI()` function checks the significance of the $\mathcal{I}$ set and outputs it to the bitstream. In addition to that, if the result is indeed a positive one, it also calls the `CodeI()` function.

Lastly, the `CodeI()` function partitions $\mathcal{I}$ into three sets $\mathcal{S}_i$ and one updated $\mathcal{I}$, as depicted in Fig. 5. Also, for each $\mathcal{S}_i$, it invokes `ProcessS(`$\mathcal{S}_i$`)` and, lastly, `ProcessI()` for the updated version of $\mathcal{I}$.

For a complete description of SPECK and all its properties, the reader is referred to [1].



(a) Group of Images (*spectral bands*)



(b) Four-dimensional vector sample $\boldsymbol{v}(m_1, m_2)$

Fig. 6. Spectrally-adjacent Group of Images (GOI) for encoding in LVQ-SPECK.

B. The LVQ-SPECK Algorithm

The compression algorithms herein proposed are variants of the two-dimensional SPECK, modified to deal with multidimensional data. In particular, the first proposed version, LVQ-SPECK [41], treats each spectral vector as a *multidimensional pixel*, following the encoding steps originally defined for SPECK.

In LVQ-SPECK, each component of this n-dimensional pixel $\boldsymbol{v}(m_1, m_2)$ is extracted from the same spatial position (m_1, m_2) in a set of n adjacent spectral bands. In particular, Fig. 6 shows how a vector sample $\boldsymbol{v}(m_1, m_2)$ is defined for a given *Group of Images* (GOI) of dimension 4. Hence, for each spatial coordinate, we have

$$\boldsymbol{v}(m_1, m_2) = (b_n(m_1, m_2), b_{n+1}(m_1, m_2), b_{n+2}(m_1, m_2), b_{n+3}(m_1, m_2)) \,, \tag{15}$$

where each component belongs to a distinct spectral band.

Amongst the necessary changes made to SPECK so that it would become a vector-based codec, we may cite the introduction of a lattice-based vector codebook, the definition of vector significance against a threshold, and the threshold scaling factor α.

Being a successive approximation based method, however, LVQ-SPECK retains those characteristics that make the class of encoders such as SPECK a very successful one, such as the embeddedness of the bitstream and its quality and rate scalability. Spatial resolution scalability is

also inherently present, since the sorting and refinement steps will still deal with blocks who are spatially confined within a subband.

LVQ-SPECK applies a 2D discrete wavelet transform to each of the scalar bands, generating a group of adjacent data sets containing transform coefficients. Even though LVQ-SPECK encodes a three-dimensional block of data, it does not use any kind of transform in the spectral direction. Therefore, the dataset to be encoded is comprised of sets of 2D wavelet transformed coefficients.

The LVQ-SPECK encoding process follows the vector-based successive approximation method presented by Eq. 12. In other words, during each encoding pass, after a vector $\boldsymbol{v}(m_1, m_2)$ is deemed significant, the approximation is done by choosing the one codeword that best represents the residual error between the original data and its current reconstructed version.

Since LVQ-SPECK deals with vector quantities, the significance measure used will now compare the norm of the vectors in the set $\mathcal{T}$ being encoded against the current threshold T_n, that is

$$\Gamma_n(\mathcal{T}) = \begin{cases} 1, & \text{if } \max\limits_{(m_1,m_2)\in\mathcal{T}} \|\boldsymbol{v}(m_1, m_2)\| \geq T_n \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

The initial encoding threshold is defined based on the largest value to be encoded,

$$v_{\max} = \max_{(m_1,m_2)\in\mathcal{X}} (\|\boldsymbol{v}(m_1, m_2)\|) \tag{17}$$

which in this case is the largest L_2-norm among all the transform vectors. However, the threshold scaling rate is no longer restricted to 0.5, which is the usual rule in scalar bitplane encoders.

The LVQ-SPECK algorithm defines the same two classes of partitioning sets, $\mathcal{S}$ and $\mathcal{I}$ (shown in Fig. 3), used to convey the significance information of a group of samples. Initially, the $\mathcal{S}$ set is defined to be the set comprised of the LL frequency subband vectors of DWT coefficients, while the $\mathcal{I}$ set accounts for all the remaining subbands.

LVQ-SPECK follows the encoding rules of the SPECK algorithm [1], with the modifications to the *Initialization* and *Quantization* passes presented in Fig. 7.

The procedures involved in the LVQ-SPECK encoding and decoding processes that are different from the original ones defined for SPECK are likewise presented in Figs. 8 and 9. Those functions involved with processing sets of type $\mathcal{I}$, as they are identical to those in SPECK, are omitted from the algorithm description. For more information, the reader is referred to [1].

Examination of the algorithm shows us that the encoding power of LVQ-SPECK stems from the fact that it sorts out those vectors with larger magnitude and immediately starts sending

`Initialization:`
Start
Apply 2D DWT to each one of the n bands to be processed
Output $T_0 = \alpha v_{\max}$
Start
Partition image transform $\mathcal{X}$ into $\mathcal{S}$ and $\mathcal{I} = \mathcal{X} - \mathcal{S}$ sets
Add $\mathcal{S}$ to LIS
Set LSP $= \emptyset$
End
End

`Quantization-pass update:`
Start
$T_{n+1} = \alpha T_n$
goto: `Sorting Pass:`
End

Fig. 7. The LVQ-SPECK Algorithm – modifications when compared to SPECK

information about their spatial location and orientation on the n-dimensional hypersphere. Subsequent passes provide refinement information, further reducing the reproduction distortion. It is also worth noticing that, as in the original (scalar) SPECK codec, the generated bitstream is still an embedded one.

C. LVQ-SPECK simulation results

The LVQ-SPECK algorithm was used to encode the AVIRIS hyperspectral *radiance* datasets Cuprite (scenes 01 and 04) and Jasper Ridge (scenes 01 and 03) [17]. Is is worth observing that the performance results for LVQ-SPECK presented in [41] were produced by encoding *reflectance* hyperspectral blocks. All datasets had their dimensions cropped to 512×512×224. The spectral bands were then grouped into n-dimensional blocks to be encoded, where n is the dimension of the lattice being used.

The DWT kernel used was the 9/7-tap DWT [42], and a 5-stage transform was applied to each spectral band. No transform was used in the spectral direction.

Bit allocation across subbands is done implicitly based on the significance of each vector being encoded. Each significance test accounts for one bit in the final bitstream and, since both four-dimensional codebooks used contain 24 vectors, in the worst case vector index transmission will

```
ProcessS(S):
Start
    Output: Γ_n(S)
    if Γ_n(S) = 1 then
        if S is a pixel v(·) then
            Output: sign of v(·)
            Output: index of codeword that better approximates v(·)
            Output: Add S to LSP
        else
            CodeS(S)
        end if
        if S ∈ LIS then
            Remove S from LIS
        end if
    else
        if S ∉ LIS then
            Add S to LIS
        end if
    end if
    return
End
```

Fig. 8. Functions used by the LVQ-SPECK Algorithm – `ProcessS(`$\mathcal{S}$`)`.

demand $\log_2 24 = 4.59$ bits during the sorting pass and $\log_2 25 = 4.64$ bits during the refinement ones (to account for the zero codeword). Similar estimates also hold for codebooks based on larger dimensional lattices.

LVQ-SPECK needs to generate only a single bitstream with the maximum desired target bit rate. During the decoding process, if a smaller target bit rate is to be used, the bitstream is truncated and decoded to that point.

Tables VI – IX present a comparison among the reconstruction results for each of the hyperspectral blocks considered, when processed by LVQ-SPECK and 3D-SPECK algorithms [16], the multi-component feature of JPEG2000 [14], and the original 2D-SPECK codec applied to each of the spectral bands individually. The figure of merit utilized here is the signal-to-quantization noise ratio (SNR), as defined in Eq. 1. For the scene 01 of both images, results for the JPEG2000 + PC (principal component) codec proposed by [28] are also presented.

```
CodeS(S):
Start
    Partition S into four equal subsets O(S)
    for all S_i ∈ O(S), i = 0, 1, 2, 3 do
        Output: Γ_n(S_i)
        if Γ_n(S_i) = 1 then
            if S_i is a pixel v(·) then
                Output: sign of S_i
                Output: index of codeword that better approximates v(·)
                Add S_i to LSP
            else
                CodeS(S_i)
            end if
        else
            Add S_i to LIS
        end if
    end for
    return
End
```

Fig. 9. Functions used by the LVQ-SPECK Algorithm – `CodeS(`$\mathcal{S}$`):`.

For simulations employing the 3D-SPECK algorithm, the implementation used was that of the QccPack software [43]. The dataset was subdivided into blocks of dimensions 512×512×16, and a 3-level 3D DWT was applied to each one of the blocks prior to the encoding phase. In the JPEG2000 case, a 2D spatial DWT was applied to each band in a group of 16, and the resulting transformed dataset was encoded using the multi-component mode present in the Kakadu software [44].

To determine the values of α used in the simulations, n consecutive spectral bands were encoded using different values of α in a given range (e.g. [0.60, 0.75] for four-dimensional lattices). The value of α which yielded the best rate-distortion performance was then used to encode the whole dataset.

The obtained results show us that the performance attained by the LVQ-SPECK algorithm is close to that of 3D-based codecs, such as 3D-SPECK, even without the use of a transform in the spectral direction. That is, in fact, quite impressive, considering that in the case of 3D algorithms, the decorrelating transform across the spectral direction has length 16, compared to vector dimensions of 4, 8 and 16 used by LVQ-SPECK.

Rate (bpp)	0.1	0.2	0.5	1.0
LVQ-SPECK D_4 shell-2	18.96	22.82	29.73	35.67
MLVQ-SPIHT	12.36	18.28	25.10	31.31

TABLE III

SNR results (in dB) for the Moffet Field scene 03 – reflectance

In particular, the difference in performance exhibited by the two versions of D_4-based shells provides us with the evidence of what a proper rotation might accomplish. The reason for that is the existence, in the D_4 shell-2 codebook, of vectors of the form $(\pm 1, \pm 1, \pm 1, \pm 1)$, which allow for simultaneous decrease in distortion for all the bands being encoded. The D_4 shell-1 codebook, on the other hand, has vectors of the form $(\pm 1, \pm 1, 0, 0)$, and therefore it is able to approximate only two components at a time.

It is also clear from Tables VI – IX that simultaneously encoding a group of spectral bands using LVQ-SPECK provides much better results than the individual compression of each one of them. For instance, for a rate of 1.0 bpp, there is a gain of about 2 dB in SNR for all the images tested, in the worst case. The best results show an improvement of approximately 10 dB over the scalar version of SPECK.

1) Comparison with MLVQ-SPIHT: In order to present a performance comparison of LVQ-SPECK and MLVQ-SPIHT [27], the *reflectance* dataset extracted from the Moffet Field scene 03 was compressed/decompressed using both algorithms. LVQ-SPECK employed the D_4 shell-2 codebook, while MLVQ-SPIHT was used with the spherical version of D_4 and the L_2-norm significance test.

Results for both encoders are presented on Table III, and they represent the average value of SNR in dB for the 204 non-zero spectral bands. It can be seen that LVQ-SPECK attains good encoding performance in the case of reflectance images as well, outperforming MLVQ-SPIHT by 4.0 dB at rates of 1.0 bpp and even larger margins for lower encoding target rates.

D. The DWP-SPECK Algorithm

In a successive approximation setting such as the one employed by LVQ-SPECK, if large energy differences exist among the spectral bands, it is expected that the amount of information needed to characterize them will vary greatly as well.

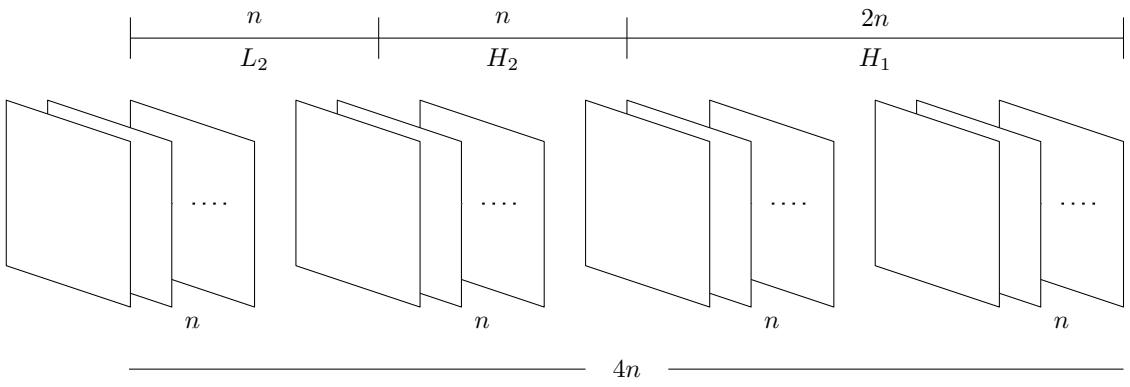


Fig. 10. DWP-SPECK Group of Bands to be encoded

In particular, those bands with small variances usually require smaller rates in order to achieve a given reproduction quality, allowing for some of the remaining bit budget to be used in the refinement of data from those spectral bands with larger variances.

Therefore an overall gain in average performance is expected if, instead of encoding a number of adjacent spectral bands equal to the codeword dimension, we opt to simultaneously encode an integer multiple of that dimension.

In this proposed vector-based version of SPECK, if the codebook in use is based on a lattice vector quantizer of dimension n, the number of bands to be encoded will be set as $4n$.

Once again a 2D DWT will be applied to each of the spectral bands separately, resulting in a dataset comprised of $4n$ transformed bands. However, in addition to applying a spatial transform, each $4n$ spectral vector will be subjected to a 1D DWT, resulting in a *discrete wavelet packet* decomposition (DWP) of the input dataset [8], [45]. Hence the denomination DWP-SPECK [46].

In DWP-SPECK each group of n bands will define a *Group of Bands* (GOB), as illustrated in Fig. 10, and during the encoding process each GOB will be treated as the single group of images was in LVQ-SPECK.

Each group of bands will have its own $\mathcal{S}$ and $\mathcal{I}$ sets, with $\mathcal{S}$ being defined as the set containing the LL subbands of that GOB and $\mathcal{I}$ as its complement in relation to that particular GOB (as in Fig. 3).

Bit allocation among the GOBs is done in a straightforward way. The encoder starts by determining which n-dimensional vector from the full dataset, i.e. the four GOBs, possesses the largest norm. The initial encoding threshold is then calculated from that norm and used to determine

vector significance according to the same principles outlined for the LVQ-SPECK algorithm.

The main difference between LVQ-SPECK and DWP-SPECK resides in the fact that initially there are four different $\mathcal{S}$ and $\mathcal{I}$ sets and that, at any time in the encoding process, the partitioning process in a given GOB does not interfere whatsoever with those of the remaining ones.

The modifications present in DWP-SPECK do not preclude this algorithm from possessing the resolution scalability feature. Higher frequency bands in the space-frequency domain may still be discarded in order to obtain a lower spatial resolution version of the original dataset.

It must be stressed, however, that the encoding threshold T_n and the threshold reduction factor α are common to all GOBs. The steps followed by the encoder can therefore be summarized as:

- Apply spatial 2D DWT to each of the $4n$ spectral bands
- Apply spectral 1D DWT to each of the $4n$-dimensional *spectral* vectors that comprise the dataset.
- Scan the dataset (now made up of wavelet packet coefficients) and determine the largest vector norm; establish initial encoding threshold T_0.
- Define initial encoding sets and process them as in LVQ-SPECK.

It should be noted that no constraints are placed in the choice of the additional DWT and, in fact, different choices for the spatial and spectral decompositions are often used. Also, while it was not the purpose of the present work to investigate that, the wavelet packet decomposition can be further subjected to an optimization procedure to maximize the coding gain obtained with its use [8].

E. DWP-SPECK Experimental Results

In simulations with the DWP-SPECK codec, we used *radiance* scenes from the AVIRIS hyperspectral images Cuprite and Jasper Ridge [17], as opposed to the results presented in [46] which were produced for *reflectance* datasets. All image blocks were cropped to 512×512×224. The spectral bands were then grouped into $4n$-dimensional blocks to be encoded.

As in the LVQ-SPECK case, the values of α used in the simulations were determined by taking n consecutive spectral bands, where n is the codebook dimension, and encoding them using different values of α. The value of α which yielded the best rate-distortion performance was then used to encode the whole dataset.

The transform kernel used was the 9/7-tap bi-orthogonal wavelet [42]. A 5-stage 2D transform was applied to each spectral band and a 2-stage 1D transform was applied in the spectral direction.

To obtain a lower rate reconstructed version of the dataset, it suffices to truncate the encoded bitstream to the desired rate. As in the original SPECK codec, if the encoded bitstream was generated with a target rate R, it contains all lower rate bitstreams whose target rates are $r \leq R$.

Tables X – XIII present reconstruction results for each of the hyperspectral blocks, when processed by DWP-SPECK, the 3D-SPECK algorithm [16], the JPEG2000 multi-component algorithm [14], and the original SPECK codec applied to each of the spectral bands individually. For the scene 01 of both images, results for the JPEG2000 + PC (principal component) codec proposed by [28] are also presented.

The figure of merit utilized here is the signal-to-quantization noise ratio (SNR), as previously defined by Eq. 1, where P_x is the average power of the original signal and MSE is the reproduction mean-squared error.

Careful observation of the presented SNR values shows that significant improvements were obtained with the introduction of both the discrete wavelet packet transform and the increased number of bands simultaneously processed.

In particular, for a rate of 1.0 bpp, it can be seen that DWP-SPECK consistently outperforms LVQ-SPECK by a margin of 2-10 dB, depending on the choice of codebook and attains performance comparable to that of 3D-SPECK. For lower rates, DWP-SPECK outperforms even the 3D codecs.

It is also interesting to notice that the results obtained by DWP-SPECK with the D_4 shell-2 codebook are consistently better than those of the LVQ-SPECK algorithm, even though both are encoding the same 16-long group of images. This can be directley traced to the DWT spectral decorrelation capabilities enjoyed by the DWP-SPECK codec, which account for a much better coding gain.

V. Codebook Reduction for LVQ-based Successive Approximation Codecs

This section will show how it is possible to condition statistics of a given codebook on the vector chosen at the previous encoding step. We start with the basic concepts, describe the implementation setup and show the obtained simulation results for the compression of hyperspectral images.

A. Basic Concepts

In successive approximation vector quantization, when a vector is chosen at a given step, some vectors in the codebook will never be chosen in the next approximation step. In the two-dimensional

i	$\boldsymbol{r}_i$	$\|\boldsymbol{r}_i\|$	T_i	$T_i \boldsymbol{c}_{k(i)}$	$\bar{\boldsymbol{x}} = \sum_{j=0}^{i} T_j \boldsymbol{c}_{k(j)}$
0	$\boldsymbol{x} = (4, 6)$	7.2111	5.0478	(0, 5.0478)	(0, 5.0478)
1	$(4, 0.9522)$	4.1118	3.5334	(3.5334, 0)	(3.5334, 5.0478)
2	$(0.4666, 0.9522)$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

TABLE IV

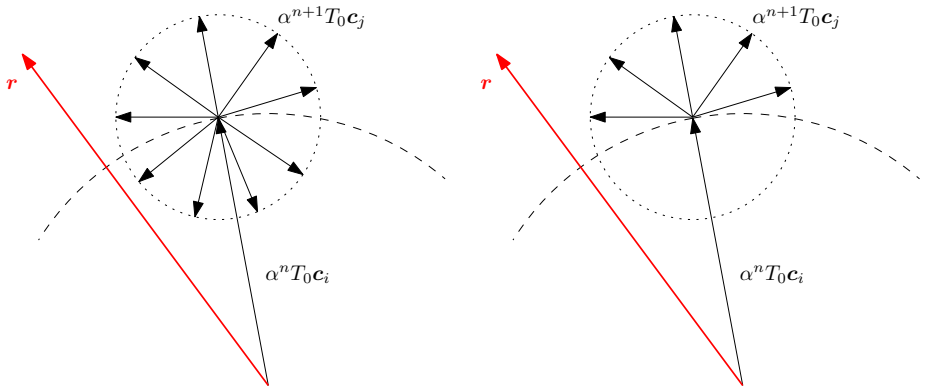
EXAMPLE OF SUCCESSIVE APPROXIMATION OF A VECTOR USING THE $\mathbb{Z}^2$ LATTICE.

case, Figs. 11(a) and 11(b) illustrate the available approximation options at each encoding pass, as depicted in Fig. 2.

In order to gain some perspective, consider the successive approximation example described in Table IV, in which the vector $\boldsymbol{x} = (4, 6)$ is to be approximated using the lattice $\mathbb{Z}^2$. The contraction rate factor $\alpha = 0.7$. It can be seen that the norm of the residue $\boldsymbol{r}_i$ decreases rapidly and that, in just two iterations, the reconstructed vector $\bar{\boldsymbol{x}}$ is a good approximation of the original data.

Looking into this with more detail, one can see that it is reasonable to assume that the choice of a codebook vector at a given step modifies the probability of the codebook vectors being chosen at the next step. In order to test this assumption, we incorporated changes in our proposed algorithms.

One of them is to calculate which vectors of a codebook are forbidden to be chosen given a choice of a codevector at a previous step. The other is to use a different probability model for the encoding of the codevectors for each choice of a codevector at the previous step. In this way, for



(a) Using the full codebook (b) Using one possible codebook reduction

Fig. 11. Two-dimensional refinement pass

θ	0°	60°	75°	90°	105°	120°	180°
D_4 shell-1	1	8	0	6	0	8	1
D_4 shell-2	1	8	0	6	0	8	1
E_8	1	56	0	126	0	56	1
Λ_{16}	1	280	1024	1710	1024	280	1

TABLE V

HISTOGRAM OF THE ANGLES BETWEEN VECTORS FOR THE TESTED LATTICES

a codebook with N vectors, we end up with N probability models.

There is no need to establish *a priori* any conditional probability models for the codevectors. Rather, we deal with this problem by, in the adaptive arithmetic coder, using a different adaptive model for each conditioning vector.

In order to evaluate the potential impact of such a proposition in the rate-distortion characteristics of our proposed method, we have computed the histograms of the angles (θ) between the vectors in the codebook, for the lattices considered in this work. The results are summarized in Table V.

By looking at Figs. 11(a) and 11(b), one could conjecture that, given a codevector chosen at a step, the one chosen at the next step cannot be at an angle larger than 90°. From that, analysis of Table V would eliminate from the next step a large number of vectors. For example, for D_4 shell-1 9 vectors would be eliminated.

However, this reasoning does not take into account that the Voronoi region of codevectors that are at an angle larger than 90° with the codeword chosen in the previous step may have a non-null intersection with the n-dimensional half-space defined by that codeword.

This, in turn, means that one has to eliminate the vectors whose Voronoi regions do not have any vector with less than 90° with the conditioning ones. Therefore, in the worst case, one would have to subtract $\theta_{\max}$ (Table II) from the angles in Table V. Unfortunately, that means that only one vector can be discarded in all cases. This vector is the one that is at 180° with the previous one, that is, its symmetrical.

Yet, since this always implies a reduction in rate, it is worthy to exclude this vector. However, it is still interesting to investigate the conditioning of the probability model of the vectors on the

previously chosen codevector. In the next section, we describe the implementation of this idea.

B. Implementation

We introduce the proposed modification in both LVQ-SPECK and DWP-SPECK by implementing context-based encoding of the vectors in the refinement pass. The basic idea is to create two distinct refinement codebooks.

One will contain codewords that, given the last approximation vector, point towards the outside of the current encoding hypersphere, as in Fig. 11(b). This reduced codebook will be named *white* codebook.

The second codebook contains the remaining codewords, with the exception of the vector that lies at an angle of 180° with the last approximation vector, that is, its symmetric. This will be the *gray* refinement codebook.

A flag (white/gray) is transmitted prior to the index of the encoded codeword to switch between both reduced codebooks.

Formally, given that the vector chosen at the $(n-1)$-th step is $\boldsymbol{c}_{k_{n-1}}$, the encoder proceeds as follows:

a) It encodes each vector with a pair (flag,index). The flag can have two values, *gray* and *white*.
   (i) The white flag indicates vectors that are pointing towards the outside part of the current encoding hypersphere.
   (ii) The gray flag indicates vectors that are pointing inwards, but whose Voronoi regions intersect the outside part of the encoding hypersphere.

b) When it sends the white flag, the codebook at pass n, C_n^w, will contain only vectors at an angle of 90° or less with vector $\boldsymbol{c}_{k_{n-1}}$.

c) When it sends the gray flag, the codebook at pass n, C_n^g, will contain only vectors at an angle larger than 90° with vector $\boldsymbol{c}_{k_{n-1}}$, discarding the one at 180°, which will never be used.

d) The encoding of the vectors in the different codebooks is conditioned on vector $\boldsymbol{c}_{k_{n-1}}$.

e) A context-based adaptive arithmetic coder is used to encode both flags and indices.

It is worth noticing that the encoding complexity of the original algorithms is not changed by the introduction of the two reduced codebooks in the refinement passes. In fact, the search for the closest refinement codeword remains unchanged. Once the reproduction codeword is found, the encoder reads its label (white/gray) and sends it to decoder as a flag.

C. Hyperspectral Imagery Encoding Results

In this section we present results of the LVQ-SPECK, DWP-SPECK using the reduced refinement codebook strategy described above.

Tables VI–IX show results for LVQ-SPECK with reduced refinement codebook. It can be seen here that consistent improvements are obtained for the D_4 and E_8 codebooks, while a slight loss exists for the Λ_{16} codebook.

It can be observed that when using the four and eight-dimensional white and gray codebooks, LVQ-SPECK is able to process a larger number of symbols than when using the regular refinement codebook. The Λ_{16} codebook, however, generates a smaller number of total coded symbols, as a result of the increased intersection between the Voronoi regions of gray codewords and the outside part of the encoding hypersphere.

It is also worth noticing that the rate and encoding path during sorting passes remain unchanged, and any changes in performance are solely due to the modified refinement procedure and its two classes of reduced codebooks. Once again, the use of an entropy encoder helps mitigate any overhead added by the transmission of the extra flag.

As an example, when processing the Cuprite scene 01 with LVQ-SPECK and the D_4 shell-1, the number of coded symbols increases from (5617585, 6088049) to (6552000, 7567059), an extra 2413425 symbols, with only 80886 refinement symbols being drawn from the gray codebook. When using the Λ_{16} codebook though, the number of total symbols decreases from (2655020, 8603699) to (2589435, 8216367), with 447280 refinement symbols coming from the gray dictionary. There is a total loss of 452917 symbols during the encoding process.

An exception occurs, however, when processing the Jasper Ridge scene 03 image. In this case, the encoder actually attains a better performance with the use of the reduced refinement codebook, for all choices of codebook. Even though, for the Λ_{16} lattice, the margin of gain is a small one, it is reflected in the number of total symbols coded, which rises from (2642241, 7569213) to (2670427, 7799275), an increase of 258248. The number of codewords from the gray codebook is 545058. This result, albeit rare, indicates that only a small portion of possible gray codewords were used by the encoder, thereby allowing the adaptive arithmetic encoder to reduce the resulting rate associated with the gray codebook.

Similar results can be observed for the DWP-SPECK encoder with reduced refinement codebook, as depicted on Tables X–XIII. It can be seen that, as in the LVQ-SPECK case, consistent improvements of approximately 1.0 dB (in some cases, close to 2.0 dB) for the D_4 and E_8 codebooks are

	Cuprite (scene 01)			
Rate (bpp)	0.1	0.2	0.5	1.0
JPEG2000+PC (from [28])	45.30		50.50	54.20
JPEG2000 MC (16-band blocks)	27.37	29.24	32.58	36.27
3D-SPECK (16-band blocks)	35.24	39.50	45.76	49.45
SPECK	27.13	28.92	32.08	35.47
LVQ-SPECK D_4 shell-1 ($\alpha = 0.67$)	28.20	30.19	33.82	37.30
LVQ-SPECK D_4 shell-2 ($\alpha = 0.69$)	30.50	33.26	38.45	44.19
LVQ-SPECK E_8 ($\alpha = 0.69$)	**32.75**	**36.21**	**42.43**	**47.36**
LVQ-SPECK Λ_{16} ($\alpha = 0.77$)	31.98	35.23	40.65	45.84
(RC)LVQ-SPECK D_4 shell-1 ($\alpha = 0.67$)	28.47	30.80	34.51	39.11
(RC)LVQ-SPECK D_4 shell-2 ($\alpha = 0.69$)	30.56	33.37	38.71	44.80
(RC)LVQ-SPECK E_8 ($\alpha = 0.69$)	**32.80**	**36.37**	**43.01**	**47.98**
(RC)LVQ-SPECK Λ_{16} ($\alpha = 0.77$)	30.93	34.64	40.25	45.21

TABLE VI

AVERAGE SNR (IN DB) FOR (RC)LVQ-SPECK – CUPRITE SCENE 01

	Cuprite (scene 04)			
Rate (bpp)	0.1	0.2	0.5	1.0
JPEG2000 MC (16-band blocks)	27.49	29.35	32.64	36.30
3D-SPECK (16-band blocks)	35.19	39.42	45.75	49.67
SPECK	27.28	29.03	32.16	35.52
LVQ-SPECK D_4 shell-1 ($\alpha = 0.67$)	28.27	30.25	33.82	37.41
LVQ-SPECK D_4 shell-2 ($\alpha = 0.69$)	30.55	33.28	38.48	44.20
LVQ-SPECK E_8 ($\alpha = 0.70$)	**32.76**	**36.21**	**42.35**	**47.36**
LVQ-SPECK Λ_{16} ($\alpha = 0.72$)	32.07	35.24	40.84	45.82
(RC)LVQ-SPECK D_4 shell-1 ($\alpha = 0.67$)	28.58	30.87	34.53	39.15
(RC)LVQ-SPECK D_4 shell-2 ($\alpha = 0.69$)	30.62	33.38	38.73	44.83
(RC)LVQ-SPECK E_8 ($\alpha = 0.70$)	**32.78**	**36.35**	**42.88**	**48.12**
(RC)LVQ-SPECK Λ_{16} ($\alpha = 0.72$)	31.08	34.58	40.72	45.78

TABLE VII

AVERAGE SNR (IN DB) FOR (RC)LVQ-SPECK – CUPRITE SCENE 04

	Jasper Ridge (scene 01)			
Rate (bpp)	0.1	0.2	0.5	1.0
JPEG2000+PC (from [28])	38.60		46.70	50.40
JPEG2000 MC (16-band blocks)	18.41	19.93	22.76	26.24
3D-SPECK (16-band blocks)	24.29	28.11	35.37	41.65
SPECK	18.14	19.59	22.25	25.38
LVQ-SPECK D_4 shell-1 ($\alpha = 0.70$)	19.12	20.81	24.02	27.88
LVQ-SPECK D_4 shell-2 ($\alpha = 0.66$)	20.97	23.37	28.52	34.63
LVQ-SPECK E_8 ($\alpha = 0.69$)	**22.76**	**26.04**	**32.46**	**38.33**
LVQ-SPECK Λ_{16} ($\alpha = 0.75$)	22.33	25.45	31.42	37.43
(RC)LVQ-SPECK D_4 shell-1 ($\alpha = 0.70$)	19.37	21.17	24.77	29.46
(RC)LVQ-SPECK D_4 shell-2 ($\alpha = 0.66$)	21.02	23.46	28.76	35.47
(RC)LVQ-SPECK E_8 ($\alpha = 0.69$)	**22.77**	**26.16**	**33.02**	**39.81**
(RC)LVQ-SPECK Λ_{16} ($\alpha = 0.75$)	21.53	24.95	31.15	36.63

TABLE VIII

AVERAGE SNR (IN DB) FOR (RC)LVQ-SPECK – JASPER RIDGE SCENE 01

	Jasper Ridge (scene 03)			
Rate (bpp)	0.1	0.2	0.5	1.0
JPEG2000 MC (16-band blocks)	28.44	29.45	31.68	34.54
3D-SPECK (16-band blocks)	24.41	28.07	35.31	41.53
SPECK	18.42	19.80	22.35	25.31
LVQ-SPECK D_4 shell-1 ($\alpha = 0.67$)	19.30	20.91	23.94	27.56
LVQ-SPECK D_4 shell-2 ($\alpha = 0.69$)	21.14	23.43	28.31	34.38
LVQ-SPECK E_8 ($\alpha = 0.70$)	**22.84**	**25.98**	**32.36**	**38.15**
LVQ-SPECK Λ_{16} ($\alpha = 0.71$)	22.47	25.45	31.20	37.03
(RC)LVQ-SPECK D_4 shell-1 ($\alpha = 0.67$)	19.42	21.14	24.64	28.89
(RC)LVQ-SPECK D_4 shell-2 ($\alpha = 0.69$)	21.17	23.48	28.49	35.07
(RC)LVQ-SPECK E_8 ($\alpha = 0.70$)	**22.86**	**26.08**	**32.82**	**39.51**
(RC)LVQ-SPECK Λ_{16} ($\alpha = 0.71$)	21.82	25.05	31.35	37.14

TABLE IX

AVERAGE SNR (IN DB) FOR (RC)LVQ-SPECK – JASPER RIDGE SCENE 03

	Cuprite (scene 01)			
Rate(bpp)	0.1	0.2	0.5	1.0
JPEG2000+PC (from [28])	45.30		50.50	54.20
JPEG2000 MC (16-band blocks)	27.37	29.24	32.58	36.27
3D-SPECK (16-band blocks)	35.24	39.50	45.76	49.45
SPECK	27.13	28.92	32.08	35.47
DWP-SPECK D_4 shell-1 ($\alpha = 0.68$)	32.90	36.28	42.16	47.35
DWP-SPECK D_4 shell-2 ($\alpha = 0.67$)	35.30	39.08	44.85	48.61
DWP-SPECK E_8 ($\alpha = 0.72$)	36.42	40.19	45.67	**49.26**
DWP-SPECK Λ_{16} ($\alpha = 0.75$)	**37.24**	**41.12**	**45.89**	49.09
(RC)DWP-SPECK D_4 shell-1 ($\alpha = 0.68$)	33.36	37.28	44.00	48.60
(RC)DWP-SPECK D_4 shell-2 ($\alpha = 0.67$)	35.61	39.80	45.67	49.21
(RC)DWP-SPECK E_8 ($\alpha = 0.72$)	**37.04**	**41.29**	**46.71**	**50.08**
(RC)DWP-SPECK Λ_{16} ($\alpha = 0.75$)	36.71	40.43	45.33	48.57

TABLE X

SNR RESULTS (IN DB) FOR (RC)DWP-SPECK – CUPRITE SCENE 01.

obtained, as well as a slight loss for the Λ_{16} codebook.

As an example, for the Cuprite scene 04 image, the number of coded symbols by DWP-SPECK increases from (2942913, 7372809) to (3419335, 7968725), a difference of 1072338, using the E_8 codebook. For the Λ_{16} codebook, however, there is a decrease from (2187099, 5975340) to (2196847, 5923730) total symbols, a loss of 41862.

Consistent behavior in terms of attained performance and number of coded symbols can be observed for all the tested datasets and codebooks.

D. Summary

In this section we presented modified versions of LVQ-SPECK and DWP-SPECK in which we use, in the refinement pass, conditioning on the previously chosen codevector.

We have shown that to create a reduced version of the codebook for successive approximation purposes is not as straightforward as it may seem, given that in an n-dimensional space, a careful examination of the volumes of Voronoi regions must be considered. In fact, given the conditioning codevector, only its symmetrical may be immediately excluded from the refinement codebook.

	Cuprite (scene 04)			
Rate(bpp)	0.1	0.2	0.5	1.0
JPEG2000 MC (16-band blocks)	27.49	29.35	32.64	36.30
3D-SPECK (16-band blocks)	35.19	39.42	45.75	49.67
SPECK	27.28	29.03	32.16	35.52
DWP-SPECK D_4 shell-1 ($\alpha = 0.69$)	33.09	36.15	42.31	47.48
DWP-SPECK D_4 shell-2 ($\alpha = 0.68$)	35.07	38.95	44.91	48.76
DWP-SPECK E_8 ($\alpha = 0.69$)	36.43	40.23	45.76	**49.43**
DWP-SPECK Λ_{16} ($\alpha = 0.70$)	**37.02**	**40.99**	**45.96**	49.33
(RC)DWP-SPECK D_4 shell-1 ($\alpha = 0.69$)	33.64	37.67	43.67	48.66
(RC)DWP-SPECK D_4 shell-2 ($\alpha = 0.68$)	35.48	39.56	45.61	49.29
(RC)DWP-SPECK E_8 ($\alpha = 0.69$)	**37.05**	**41.24**	**46.71**	**50.27**
(RC)DWP-SPECK Λ_{16} ($\alpha = 0.70$)	36.90	40.99	46.05	49.25

TABLE XI

SNR results (in dB) for (RC)DWP-SPECK – Cuprite scene 04.

	Jasper Ridge (scene 01)			
Rate(bpp)	0.1	0.2	0.5	1.0
JPEG2000+PC (from [28])	38.60		46.70	50.40
JPEG2000 MC (16-band blocks)	18.41	19.93	22.76	26.24
3D-SPECK (16-band blocks)	24.29	28.11	35.37	41.65
SPECK	18.14	19.59	22.25	25.38
DWP-SPECK D_4 shell-1 ($\alpha = 0.68$)	23.15	26.39	33.00	38.80
DWP-SPECK D_4 shell-2 ($\alpha = 0.68$)	24.40	27.99	34.16	39.91
DWP-SPECK E_8 ($\alpha = 0.73$)	**26.05**	**29.99**	36.28	41.35
DWP-SPECK Λ_{16} ($\alpha = 0.77$)	25.98	29.83	**36.32**	**41.48**
(RC)DWP-SPECK D_4 shell-1 ($\alpha = 0.68$)	23.76	27.05	34.40	40.25
(RC)DWP-SPECK D_4 shell-2 ($\alpha = 0.68$)	24.75	28.44	35.22	40.79
(RC)DWP-SPECK E_8 ($\alpha = 0.73$)	**26.55**	**30.74**	**37.48**	**42.67**
(RC)DWP-SPECK Λ_{16} ($\alpha = 0.77$)	25.00	28.74	35.10	40.20

TABLE XII

SNR results (in dB) for (RC)DWP-SPECK – Jasper Ridge scene 01.

	Jasper Ridge (scene 03)			
Rate(bpp)	0.1	0.2	0.5	1.0
JPEG2000 MC (16-band blocks)	28.44	29.45	31.68	34.54
3D-SPECK (16-band blocks)	24.41	28.07	35.31	41.53
SPECK	18.42	19.80	22.35	25.31
DWP-SPECK D_4 shell-1 ($\alpha = 0.68$)	23.23	26.44	32.55	38.71
DWP-SPECK D_4 shell-2 ($\alpha = 0.70$)	24.47	27.94	34.10	39.76
DWP-SPECK E_8 ($\alpha = 0.69$)	**26.10**	**30.08**	**36.60**	**41.46**
DWP-SPECK Λ_{16} ($\alpha = 0.76$)	25.99	30.06	36.29	41.41
(RC)DWP-SPECK D_4 shell-1 ($\alpha = 0.68$)	23.59	27.42	34.73	40.23
(RC)DWP-SPECK D_4 shell-2 ($\alpha = 0.70$)	24.70	28.41	35.32	40.92
(RC)DWP-SPECK E_8 ($\alpha = 0.69$)	**26.37**	**30.61**	**37.41**	**42.67**
(RC)DWP-SPECK Λ_{16} ($\alpha = 0.76$)	24.61	28.59	34.81	39.96

TABLE XIII

SNR results (in dB) for (RC)DWP-SPECK –Jasper Ridge scene 03.

However, a suitable solution that allows for a considerable reduction – the use of two distinct classes of reduced codebooks – was proposed and the results have shown, in all cases, a consistent improvement of up to 2.0 dB on the rate-distortion results for the D_4 and E_8 codebooks, with a slight loss for the Λ_{16} codebook.

VI. Conclusions

In this article we presented two vector-based extensions of the state-of-the-art codec SPECK. These extensions were developed to compress volumetric datasets, such as those resulting from remote sensing applications in the form of hyperspectral images.

Based on the encoding process defined for SPECK, modifications were introduced to simultaneously process a number n of spectral bands, where n corresponds to the dimension of the codewords that form the codec dictionary.

Considering the approximation method chosen, orientation codebooks were defined based on sets of vectors extracted from those n-dimensional lattices that are known to possess the smallest values of $\theta_{\max}$ and, therefore, the best covering properties in their dimensions.

The resulting algorithm, termed LVQ-SPECK, maintains all the desired characteristics of the

original SPECK codec, such as embeddedness, SNR scalability, to name a few.

It was shown that the results obtained by LVQ-SPECK in the compression of hyperspectral images are quite competitive. Furthermore, an additional contribution of our work was to show that a suitable rotation of the codebook in use may yield results close to those of 3D codecs, that employ a 3D discrete wavelet transform prior to the encoding phase.

Another proposed innovation is a second vector-based version of SPECK, termed DWP-SPECK, that uses a discrete wavelet packet transform and simultaneously processes a larger number of spectral bands – namely, $4n$ bands at a time – to further explore the encoder's capabilities of fast converging to points of high significance in the process. DWP-SPECK encoding results are very good, outperforming in most cases other state-of-the-art codecs.

Lastly, we have introduced a new technique in which we use statistical conditioning to encode the vectors. The conditioning is based on the previously encoded vector. It shows a consistent rate-distortion improvement for all images and most codebooks.

Based on the obtained results, we conclude that both the presented contributions are quite promising, helping extend a little further the existing limits in the compression of multidimensional datasets.

References

[1] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, low-complexity image coding with a set-partitioning embedded block coder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, pp. 1219–1235, 2004.

[2] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, 1996.

[3] D. S. Taubman, "High performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, July 2000.

[4] T. R. Fischer, "A pyramid vector quantizer," *IEEE Transactions on Information Theory*, vol. IT-32, pp. 568–583, 1986.

[5] M. Barlaud, P. Sole, T. Gaidon, M. Antonini, and P. Mathieu, "Pyramidal lattice vector quantization for multiscale image coding," *IEEE Transactions on Image Processing*, vol. 3, no. 4, pp. 367–381, July 1994.

[6] W. Li, H. Cao, S. Li, F. Ling, S. Segan, H. Sun, J. Wus, and Y.-Q. Zhang, "A video coding algorithm using vector-based techniques," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 7, no. 1, pp. 146–157, feb 1997.

[7] I. Daubechies, *Ten Lectures on Wavelets*. SIAM: Society for Industrial and Applied Mathematics, 1992.

[8] S. Mallat, *A Wavelet Tour of Signal Processing, Second Edition (Wavelet Analysis & Its Applications)*, 2nd ed. Academic Press, 1999.

[9] J. W. Woods, *Subband Image Coding*. Kluwer Academic Publishers, 1990.

[10] K. Sayood, *Introduction to Data Compression*, 2nd ed. Morgan Kaufmann, 2000.

[11] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 41, no. 12, pp. 3445–3462, December 1993.

[12] 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, June 1987.

[13] A. Islam and W. A. Pearlman, "An embedded and efficient low-complexity hierarchical image coder," in *Visual Communications and Image Processing*, ser. Proceedings of SPIE, vol. 3653, January 1999, pp. 294–305.

[14] D. S. Taubman and M. W. Marcellin, *JPEG2000 Image Compression: Fundamentals, Standards and Practice*, ser. (The International Series in Engineering and Computer Science). Kluwer Academic Publishers, 2002.

[15] B.-J. Kim, Z. Xiong, and W. A. Pearlman, "Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-D SPIHT)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 8, pp. 1374–1387, Dec 2000.

[16] X. Tang and W. A. Pearlman, *Three-Dimensional Wavelet-Based Compression of Hyperspectral Images*. Springer, 2006, ch. 10 in Hyperspectral Data Compression.

[17] "The AVIRIS Project Homepage," http://aviris.jpl.nasa.gov, date Last Accessed, 09/10/2010.

[18] A. Kiely and M. Klimesh, "Exploiting calibration-induced artifacts in lossless compression of hyperspectral imagery," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 47, no. 8, pp. 2672–2678, aug. 2009.

[19] B. Penna, T. Tillo, E. Magli, and G. Olmo, "Transform coding techniques for lossy hyperspectral data compression," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 45, no. 5, pp. 1408–1421, May 2007.

[20] S.-E. Qian, "Study of hyperspectral and multispectral image compression using vector quantization in development of ccsds international standards," L. Bruzzone, C. Notarnicola, and F. Posa, Eds., vol. 7477, no. 1. SPIE, 2009, p. 74770O. [Online]. Available: http://link.aip.org/link/?PSI/7477/74770O/1

[21] I. Blanes and J. Serra-Sagrista, "Cost and scalability improvements to the karhunen-loêve transform for remote-sensing image coding," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 48, no. 7, pp. 2854–2863, july 2010.

[22] J. E. Fowler and J. T. Rucker, *Three-dimensional wavelet-based compression of hyperspectral imagery*. John Wiley & Sons, 2007, ch. 14.

[23] G. Motta, F. Rizzo, and J. A. Storer, "Compression of Hyperspectral Imagery," in *Proceedings of the Data Compression Conference*, March 2003, pp. 333–342.

[24] ——, *Locally Optimal Partitioned Vector Quantization of Hyperspectral Data*. Springer, 2006, ch. 5 in Hyperspectral Data Compression.

[25] X. Tang, W. A. Pearlman, and J. W. Modestino, "Hyperspectral image compression using three-dimensional image coding," in *Electronic Imaging*, ser. Proceedings of the SPIE, vol. 5022, Jan. 2003.

[26] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1991.

[27] Y. Liu and W. Pearlman, "Multistage lattice vector quantization for hyperspectral image compression," in *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, Nov. 2007, pp. 930–934.

[28] Q. Du and J. E. Fowler, "Hyperspectral image compression using jpeg2000 and principal component analysis," *IEEE Geoscience and Remote Sensing Letters*, vol. 4, no. 2, pp. 201–205, Apr. 2007.

[29] N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, ser. (Prentice-Hall Signal Processing Series). Prentice Hall, 1984.

[30] Z. Wang and A. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *Signal Processing Magazine, IEEE*, vol. 26, no. 1, pp. 98–117, Jan. 2009.

[31] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups (Grundlehren der mathematischen Wissenschaften)*, 3rd ed. Springer, 1998.

[32] L. Lovisolo, E. A. B. da Silva, and P. S. R. Diniz, "On the statistics of matching pursuits angles," *Signal Processing*, vol. 90, no. 12, pp. 3164–3184, December 2010.

[33] G. Davis, "Adaptive nonlinear approximations," Ph.D. dissertation, New York University, 1994.

[34] E. A. B. da Silva and M. Craizer, "Generalized bit-planes for embedded codes," in *Proceedings of the International Conference on Image Processing*, vol. 2, 1998, pp. 317–321.

[35] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. COM-28, no. 1, pp. 84–95, 1980.

[36] C.-C. Chao and R. M. Gray, "Image compression with a vector speck algorithm," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 2, May 2006.

[37] E. A. B. da Silva, "Wavelet transforms for image coding," Ph.D. dissertation, University of Essex, 1995.

[38] E. A. B. da Silva, D. G. Sampson, and M. Ghanbari, "A successive approximation vector quantizer for wavelet transform image coding," *IEEE Transactions on Image Processing*, vol. 5, pp. 299–310, 1996.

[39] D. Mukherjee and S. K. Mitra, "Successive refinement lattice vector quantization," *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1337–1348, Dec. 2002.

[40] ——, "Vector SPIHT for embedded wavelet video and image coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 3, pp. 231–246, Mar. 2003.

[41] A. J. S. Dutra, W. A. Pearlman, and E. A. B. da Silva, "Compression of Hyperspectral Images with LVQ-SPECK," in *Proceedings of the Data Compression Conference*, Mar. 2008.

[42] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Transactions on Image Processing*, vol. 1, no. 2, pp. 205–220, April 1992.

[43] J. E. Fowler, "QccPack software," http://qccpack.sourceforge.net. Date Last Accessed, 02/10/2010.

[44] D. S. Taubman, "Kakadu software," http://www.kakadusoftware.com. Date last accessed, 02/10/2010.

[45] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*, 1st ed. Prentice Hall PTR, 1995.

[46] A. J. S. Dutra, W. A. Pearlman, and E. A. B. da Silva, "Hyperspectral Image Compression using LVQ-SPECK," in *Proceedings of the International Symposium on Circuits and Systems*, 2008.