# Three-Dimensional Wavelet-Based Compression of Hyperspectral Images *

Xiaoli Tang and William A. Pearlman
Center for Image Processing Research
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

**Abstract**

Hyperspectral images may be treated as a three-dimensional data set for the purposes of compression. Here we present some compression techniques based on a three-dimensional wavelet transform that produce compressed bit streams with many useful properties. These properties are progressive quality encoding and decoding, progressive lossy-to-lossless encoding, and progressive resolution decoding. We feature an embedded, block-based, image coding algorithm of low complexity, called SPECK (Set Partitioning Embedded bloCK), that has been proposed originally for single images and is modified and extended to three dimensions. The resultant algorithm, Three-Dimensional Set Partitioning Embedded bloCK (3D-SPECK), efficiently encodes 3D volumetric image data by exploiting the dependencies in all dimensions. We describe the use of this coding algorithm in two implementations, first in a purely quality or rate scalable mode and secondly in a resolution scalable mode. We utilize both integer and floating point wavelet transforms, whereby the former one enables lossy and lossless decompression from the same bit stream, and the latter one achieves better performance in lossy compression. The structure of hyperspectral images reveals spectral responses that would seem ideal candidates for compression by 3D-SPECK. We demonstrate that 3D-SPECK, a wavelet domain algorithm, like other time domain algorithms, can preserve spectral profiles

* This work was performed at Rensselaer Polytechnic Institute and was supported in part by National Science Foundation Grant No. EEC-981276. The government has certain rights in this material.

well. Compared with the lossless version of the benchmark JPEG2000 (multi-component), the 3D-SPECK lossless algorithm produces average of 3.0% decrease in compressed file size for Airborne Visible Infrared Imaging Spectrometer images, the typical hyperspectral imagery. We also conduct comparisons of the lossy implementation with other the state-of-the-art algorithms such as Three-Dimensional Set Partitioning In Hierarchical Trees (3D-SPIHT) and JPEG2000. We conclude that this algorithm, in addition to being very flexible, retains all the desirable features of these algorithms and is highly competitive to 3D-SPIHT and better than JPEG2000 in compression efficiency.

# 1 Introduction

Hyperspectral imaging is a powerful technique and has been widely used in a large number of applications, such as detection and identification of the surface and atmospheric constituents present, analysis of soil type, monitoring agriculture and forest status, environmental studies, and military surveillance. Hyperspectral images are generated by collecting hundreds of narrow and contiguous spectral bands of data such that a complete reflectance spectrum can be obtained for each point in the region being viewed by the instrument. However, at the time we gain high resolution spectrum information, we generate massively large image data sets. Access and transport of these data sets will stress existing processing, storage and transmission capabilities. As an example, the Airborne Visible InfraRed Imaging Spectrometer (AVIRIS) instrument, a typical hyperspectral imaging system, can yield about 16 Gigabytes of data per day. Therefore, efficient compression should be applied to these data sets before storage and transmission [19].

However, utilization of the data in compressed form can often be inconvenient and intractable, if it requires full decompression. One would like the bit stream to have properties of scalability and random access. There are two types of scalability of interest here - rate or quality scalability and resolution scalability. Rate scalability means that a portion of the bit stream can be decoded to provide a reconstruction at lower rate or quality. That would allow faster or lower bandwidth transmission or a quick look at the entire data set at lower quality. Resolution scalabilty would permit decoding at reduced resolution from a portion of the compressed bit stream. These scalability properties enable transmission and retrieval that are progressive by quality or resolution.

The compression schemes used in the data sets can be broadly classified

as lossless and lossy techniques. As the name suggests, lossless compression (lossless coding) reduces the redundancy of data sets without losing any information. The original images can be reconstructed exactly. This is a reversible process, and usually lossless compression can provide a compression ratio of about 2∼3:1. On the other hand, if we are willing to lose some information, we can gain significantly higher compression ratio than that of lossless compression. This is so-called lossy compression (lossy coding). Many analysis hyperspectral image applications, like common classification tools or feature extractions, can perform reliably [10][21] on images compressed to very low bit rates. Visualization applications, such as image browsing, are able to function successfully with lossy representations as well. However, hyperspectral data is very different from other remote sensing images, such as gray-level PAN images, multi-spectral images, SAR images, etc. Hyperspectral data carry rich information in the spectral domain. A ground sample point in a hyperspectral data set has a distinct spectral profile, which is the fingerprint information of the point. Some hyperspectral data users will rely on the spectrum of each point to create application products using their remote sensing algorithms. When we compress hyperspectral images, we would like to preserve the important spectral profiles. Furthermore, given the extraordinary expense of acquiring hyperspectral imagery, it makes more sense to require lossless coding for archival applications. Therefore, in this study, we present both lossless and lossy compression for hyperspectral images.

Some Vector Quantization (VQ) based algorithms were proposed for lossy or lossless hyperspectral image compressions. Ryan and Arnold [23] proposed mean-normalized vector quantization (M-NVQ) for lossless AVIRIS compression. The sample mean is scalar quantized before subtraction from the input vector so that any error in quantization of the mean is incorporated into the error vector. Each block of the image is converted into a vector with zero mean and unit standard variation. The mean and variance are scalar quantized and the input vector is vector quantized for transmission. M-NVQ results compare favorably with other compression techniques. Motta *et al.* [18] proposed a VQ based algorithm that involved locally optimal design of a partitioned vector quantizer for the encoding of source vectors drawn from hyperspectral images. Pickering and Ryan [22] jointly optimized spatial M-NVQ and spectral Discrete Cosine Transform (DCT) to produce compression ratios significantly better than those obtained by the optimized spatial M-NVQ technique alone. Pickering and Ryan's new technique can be applied on both lossless and lossy hyperspec-

tral compression. An end-use remote sensing application, maximum likelihood classification (MLC), was used to test the efficiency of the algorithm. Results shown that distortions in the data caused by the compression process result in only minor losses in classification accuracy. Other than VQ based methods, Harsanyi and Chang [8] applied Principle Component Analysis (PCA) on hyperspectral images to simultaneously reduce the data dimensionality, suppress undesired or interfering spectral signature, and classify the spectral signature of interest. This approach is applicable to both spectrally pure as well as mixed pixels. A training sequence based entropy constrained predictive trellis coded quantization scheme was also proposed recently by Abousleman *et al.* [1] for hyperspectral image compression. All these algorithms have promising performance on hyperspectral image compression. However, none of them generates embedded bit stream, and therefore cannot provide progressive transmission.

To incorporate the embedded requirement and maintain other compression performances, many promising image compression algorithms based on wavelet transform [17] were proposed recently. They are simple, efficient and have been widely used in many applications. One of them is Shapiro's Embedded Zerotree Wavelet (EZW) [28]. Said and Pearlman [26] refined and extended EZW subsequently to SPIHT. Islam and Pearlman [11, 12] proposed another low complexity image encoder with similar features – Set Partitioned Embedded bloCK (SPECK). Related in various degrees to these earlier works on scalable image compression, the EBCOT [30] (adopted as the basis for the JPEG2000 image compression standard) algorithm also uses a wavelet transform to generate the subband samples which are to be quantized and coded. EBCOT stands for Embedded Block Coding with Optimized Truncation, which identifies some of the major contributions of the algorithm. It is resolution and SNR scalable and has the random access property. EBCOT partitions each subband into relatively small blocks (typically, $32 \times 32$ or $64 \times 64$ pixels are used), and generates a separate highly scalable (or embedded) bit stream for each block. The bit streams can be independently truncated to a set of rates calculated by a bit allocation algorithm and interleaved to achieve embedding.

The state-of-the-art encoder, SPIHT, has many attractive properties. It is an efficient embedded technique. The original SPIHT was proposed for 2-dimensional image compression, and it has been extended to 3D applications by Kim and Pearlman [14]. 3D-SPIHT is the modern-day benchmark for three dimensional image compression. It has been applied on multispectral image compression by Dragotti *et al.* [6]. They use vector quantization (VQ) and

Karhunen-Loéve transform (KLT) on the spectral dimension to explore the correlation between multispectral bands. In the spatial domain, they use discrete wavelet transform, and the 3D-SPIHT sorting algorithm is applied on the transformed coefficients. Dragotti *et al.*'s algorithms are comparable to 3D-SPIHT in multispectral image compression in rate distortion performance. Applying KLT on the spectral domain of multispectral images is acceptable because multispectral images only have a small number of bands. However, for hyperspectral imagery such as AVIRIS, computing KLT for 224 bands is usually not practical. Fry [7] adopts 3D-SPIHT directly on hyperspectral image compression. His work demonstrates the speed and computational complexity advantages of 3D-SPIHT. Results show that 3D-DWT is a fast and efficient means to exploit the correlations between hyperspectral bands. We show in this study that the lossy version of 3D-SPIHT also guarantees to preserve the spectral information for remote sensing applications.

The EBCOT algorithm has also been extended to 3D applications. Annex N of Part II of JPEG2000 standard [2] is for multi-component imagery compression. 3D-DWT is applied to decorrelate the separate components for multi-component images. Extended JPEG2000 partitioning and sorting algorithm to 3D sources is used to generate the embedded bit streams. JPEG2000 multi-component is a good candidate for hyperspectral image compression. We will compare our algorithm to multi-component JPEG2000 in this study. Three Dimensional Cube Splitting EBCOT (3D CS-EBCOT) [27] initially partitions the wavelet coefficient prism into equally sized small code cubes of $64 \times 64 \times 64$ elements. The cube splitting technique is applied on each code cube to generate separate scalable bit streams. Like EBCOT, the bit streams may be independently truncated to any of a collection of different lengths to optimize the rate distortion criteria. Xu *et al.* [35] used a different method to extend EBCOT to video coding – Three-Dimensional Embedded Subband Coding with Optimized Truncation (3-D ESCOT). They treat each subband as a code cube and generate embedded bit streams for each code cube independently by using fractional bit-plane coding. Candidate truncation points are formed at the end of each fractional bit-plane.

A recently proposed 2D embedded wavelet based coder, tarp coder [29] has also been extended to 3D for hyperspectral image compression [31]. Tarp filtering operation is employed to estimate the probability of coefficient significance for arithmetic coder. 3D tarp coder is comparable to 3D-SPIHT and JPEG2000 multi-component.

In this study, we extend SPECK to 3D sources such as hyperspectral images. For an image sequence, 3D-DWT is applied to obtain a wavelet coefficient prism. Since our applications are hyperspectral images, there is not motion, but tight statistical dependency along the wavelength axis of this prism. Therefore, 3D-DWT can exploit the consequent correlation along the wavelength axis, as well as along the spatial axes. To start the algorithm, the wavelet coefficient prism is partitioned into small (three-dimensional) code blocks[1] with different sizes, and each subband is treated as a code block. Next, an extended and modified version of the SPECK sorting algorithm is applied to these code blocks to sort the significance of pixels. A block splitting algorithm similar to the cube splitting algorithm of 3D CS-EBCOT is used on the individual code blocks to test their significance. If a code block contains significant coefficients, it is split into several smaller sub-blocks. The descendant "significant" blocks are then further split until the significant coefficients are isolated. This block splitting algorithm can zoom in quickly to areas of high energy and code them first and therefore can exploit the presence of significant high frequency intra-band components. 3D-SPECK exploits detailed underlying physical modeling properties of hyperspectral images.

This paper is organized as following: We will first briefly review the discrete wavelet transform, and the SPIHT and SPECK algorithms in Section II. The proposed technique is described in detail in Section III. Section IV presents experimental results, and Section V concludes the paper.

# 2 MATERIAL AND METHODS

## 2.1 Discrete Wavelet Analysis

Wavelet transform coding provides many attractive advantages over other transform methods. This motivates intense research on this area. Many widely known coding schemes such as EZW, SPIHT, SPECK and EBCOT are all wavelet based schemes. The 9/7 tap biorthogonal filters [5], which produce floating point wavelet coefficients, are widely used in image compression techniques [26][11][12][14] to generate a wavelet transform. This transform has proved to provide excellent performance for image compression. Hence, in this study, 9/7 tap biorthogonal filters will provide the transform for our lossy

[1] It is customary to call the unit to be coded a block. This terminology is adopted here for the three-dimensional blocks in the shape of rectangular prisms.

implementation of the new algorithm. For lossless implementations, the S+P [25] filter will be utilized.

The most frequently used wavelet decomposition structure is dyadic decomposition [17]. All coding algorithms mentioned above support dyadic decomposition. Occasionally, we need the wavelet packet decomposition which provides a richer range of possibilities for signal analysis. As hyperspectral data carries rich information in the spectral domain, one may want to apply a wavelet packet instead of a dyadic wavelet transform on the spectral domain of the data. We tested both transforms on the spectral domain for our algorithm. For floating point filter implementation, the experiments show that compared to the dyadic transform, the wavelet packet transform on spectral domain yields average of 2% increase in compressed file size for AVIRIS data for lossless compression and average of 0.36 dB lower SNR for lossy compression at rate 0.1 bits per pixel per band (bpppb). Therefore, the typical dyadic decomposition structure will be used for the floating point filter implementation. For integer filters, however, the dyadic structure provides a transform that is not unitary, so we need to apply a wavelet packet structure and scaling (by bit shifts) of the wavelet coefficients to make the transform unitary. More specifically, for spatial axes, we still maintain the 2D dyadic wavelet transform for each slice with scaling of the integer wavelet coefficients; for spectral axis, we use a 1D packet structure, and again with scaling of the integer wavelet coefficients. This packet structure and scaling make the transform approximately unitary.

## 2.2 The SPIHT Algorithm

Both SPIHT and SPECK algorithms are highly ef ficient encoders. They are low in complexity, fast in encoding and decoding, and are fully embedded.

The SPIHT algorithm is the modern-day benchmark. It is essentially a wavelet transform-based embedded bit-plane encoding technique. SPIHT partitions transformed coefficients into spatial orientation tree sets (Fig. 1) based on the structure of the multi-resolution wavelet decomposition [26].

A transformed coefficient with larger magnitude has larger information content and therefore should be transmitted first [26][28]. SPIHT sorts coefficients and sends them in order of decreasing magnitude. As an example, if the largest coefficient magnitude in the source is 100, SPIHT begins sorting from the root coefficients to their descendants (as shown in Fig. 1, pixels in the root

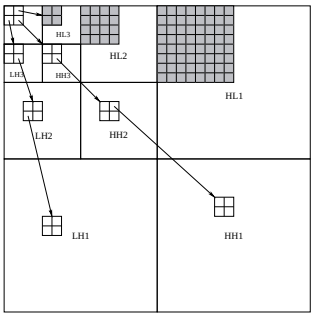


Figure 1: 2D Spatial orientation tree

can either have no or four direct descendants) to find out significant coefficients against the highest bit plane with $n = \lfloor \log_2(\max_{(i,j)} |c_{i,j}|) \rfloor = \lfloor \log_2 |100| \rfloor = 6$, where $c_{i,j}$ stands for the coefficient located on the coordinate $(i, j)$. In other words, those coefficients with magnitudes greater than or equal to $2^6$ are said to be significant with respect to $n = 6$ and will be transmitted. After finishing the transmission of all significant pixels with respect to $2^6$, SPIHT decreases $n$ by one ($n = n - 1 = 5$) to test the significance of the next lower bit plane. It then finds and transmits coefficients with $2^5 \leq |c_{i,j}| < 2^6$. Each time (except the first time) SPIHT finishes sorting one bit plane, it checks whether it found any significant coefficients on the higher planes. If so, SPIHT outputs the $n^{th}$ most significant bits of those coefficients. This process will be repeated recursively until either the desired rate is reached or all coefficients have been transmitted. As a result, SPIHT generates bit streams that are progressively refined.

One of the main features of SPIHT is that the encoder and decoder have the same sorting algorithm. Therefore, the encoder does not need to transmit the way it sorts coefficients. The decoder can recover the ordering information from the execution path, which is put into the bit-stream through the binary outputs of the significance tests.

## 2.3 The SPECK Algorithm

The SPECK algorithm has its roots primarily in the ideas developed in SPIHT and therefore has many features and properties similar to SPIHT. These two algorithms are both wavelet based techniques, and the transformed images all have hierarchical pyramidal structures. Their difference lies in the way of par-

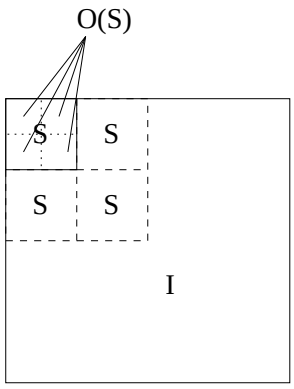


Figure 2: Set Partitioning rules for SPECK algorithm

titioning the wavelet coefficients. As stated above, SPIHT partitions wavelet coefficients into spatial orientation tree sets and sorts coefficients along the trees according to the results of significance tests. On the other hand, SPECK partitions wavelet coefficients into blocks and sorts coefficients by using the quadtree partitioning algorithm to be described.

As shown by solid lines in Figure 2, SPECK starts by partitioning the image of transformed coefficients into two sets: the set $\mathcal{S}$ which is the root (the topmost band) of the pyramid, and set $\mathcal{I}$ which is everything that is left of the image after taking out the root. Then, SPECK sorts the coefficients by testing the significance of set $\mathcal{S}$ first. The set is declared significant if there is at least one significant coefficient in this set. If $\mathcal{S}$ is found to be significant with respect to the current bit plane $n$ (the highest "1" is in this plane), it will be partitioned into four subsets $\mathcal{O}(\mathcal{S})$ (dotted lines in Fig. 2), with each subset having approximately one-fourth the size of the parent set $\mathcal{S}$. This procedure is called quadtree partitioning. Then, SPECK treats each of these four subsets as type $\mathcal{S}$ set and applies the quadtree partitioning recursively to each of them until significant pixels are located. After finishing testing of the type $\mathcal{S}$ sets, the octave band partitioning is applied to test type $\mathcal{I}$ sets. More specifically, if $\mathcal{I}$ set is found to be significant, it will be partitioned into three type $\mathcal{S}$ sets and one type $\mathcal{I}$ set as shown in Fig. 2 with dashed lines. The quadtree partitioning and octave band partitioning will be applied to these new sets respectively until significant pixels are isolated. Both quadtree and octave band partitioning algorithms can zoom in quickly to areas of high energy and therefore code them

first. By the end of the first pass, many type $\mathcal{S}$ sets of varying sizes were generated, and in the next pass, SPECK will check their significance against the next bit plane according to their sizes. This process will be repeated to test the LIS in the same way at the next lower threshold until either the desired rate is reached or all coefficients have been transmitted.

# 3 The 3D-SPECK Algorithm

In this section, we present the 3D-SPECK coding algorithm. We describe the set-up and terminology first, followed by the main body of the algorithm, some detailed discussions, and the general 3D integer wavelet packet transform structure implementation that allows bit shifting of wavelet coefficients to approximate a 3D unitary transformation.

## 3.1 Set-up and Terminology

Consider a hyperspectral image sequence which has been adequately transformed using the discrete wavelet transform (can be either an integer or floating point wavelet transform). The transformed image sequence is said to exhibit a hierarchical pyramidal structure defined by the levels of decomposition, with the topmost level being the root. Figure 3 illustrates such a structure with three-level decomposition. The finest pixels lie at the bottom level of the pyramid while the coarsest pixels lie at the top (root) level. The image sequence is represented by an indexed set of transformed coefficients $c_{i,j,k}$, located at pixel position $(i, j, k)$ in the transformed image sequence.

Pixels are grouped together in sets which comprise regions in the transformed images. Unlike 2D-SPECK, 3D-SPECK has only one type of set: $\mathcal{S}$ set. We say a set $\mathcal{S}$ is significant with respect to $n$, if

$$\max_{(i,j,k)\in\mathcal{S}} |c_{i,j,k}| \geq 2^n \tag{1}$$

Where $c_{i,j,k}$ denotes the transformed coefficients at coordinate $(i, j, k)$. Otherwise it is insignificant. For convenience, we can define the significance function of a set $\mathcal{S}$ as:

$$\Gamma_n(\mathcal{S}) = \begin{cases} 1 & : \text{ if } 2^n \leq \max_{(i,j,k)\in\mathcal{S}} |c_{i,j,k}| < 2^{n+1} \\ 0 & : \text{ else} \end{cases} \tag{2}$$

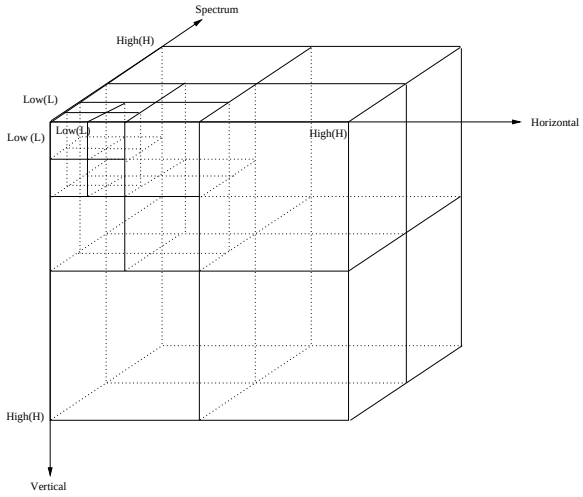


Figure 3: Structure for 3D-SPECK.

3D-SPECK makes use of rectangular prisms in the wavelet transform. Each subband in the pyramidal structure is treated as a code block or prism, henceforth referred to as sets $\mathcal{S}$, and can be of varying dimensions. The dimension of a set $\mathcal{S}$ depends on the dimension of the original images and the subband level of the pyramidal structure at which the set lies. We define the size of a set to be the number of elements in the set.

3D-SPECK maintains two linked lists:

- **LIS** – List of Insignificant Sets. This list contains $\mathcal{S}$ sets of varying sizes.
- **LSP** – List of Significant Pixels. This list contains pixels that have been found significant against a certain threshold $n$.

## 3.2 The Algorithm

Having set up and defined the terminology used in the 3D-SPECK coding method, we are now in a position to understand the main body of the actual algorithm.

The main body of 3D-SPECK consists of four steps: the initialization step; the sorting pass; the refinement pass; and the quantization step. These steps

call two functions, ProcessS() and CodeS(), which are described in detail in the following.

The algorithm starts by adding all sets $\mathcal{S}$ to the LIS.

1. **Initialization**
   - Output $n = \lfloor \log_2(\max \mid c_{i,j,k} \mid) \rfloor$
   - Set LSP = $\emptyset$
   - Set LIS = {all subbands of transformed images of wavelet coefficients }
2. **Sorting Pass**

   In increasing order of size of sets, for each set $\mathcal{S} \in$ LIS, ProcessS($\mathcal{S}$)

   ProcessS($\mathcal{S}$)

   {
   - Output $\Gamma_n(\mathcal{S})$ (Whether the set is significant respect to current $n$ or not)
   - if $\Gamma_n(\mathcal{S})$ = 1
     - if $\mathcal{S}$ is a pixel, output sign of $\mathcal{S}$ and add $\mathcal{S}$ to LSP
     - else CodeS($\mathcal{S}$)
     - if $\mathcal{S} \in$ LIS, remove $\mathcal{S}$ from LIS

   }

   CodeS($\mathcal{S}$)

   {
   - Partition $\mathcal{S}$ into eight approximately equal subsets $\mathcal{O}(\mathcal{S})$. For the situation that the original set size is odd $\times$ odd $\times$ odd, we can partition this kind of sets into different but approximately equal sizes of subsets (see Fig 4). For the situation that the size of the third dimension of the set is 1, we can partition the set into 4 approximately equal sizes of subsets.
   - For each $\mathcal{O}(\mathcal{S})$
     - Output $\Gamma_n(\mathcal{O}(\mathcal{S}))$

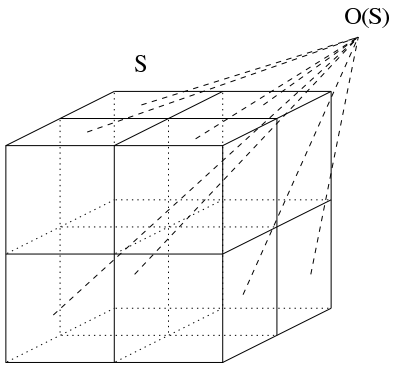


Figure 4: Partitioning of set $\mathcal{S}$

- if $\Gamma_n(\mathcal{O}(\mathcal{S}))$ = 1
  - if $\mathcal{O}(\mathcal{S})$ is a pixel, output sign of $\mathcal{O}(\mathcal{S})$ and add $\mathcal{O}(\mathcal{S})$ to LSP
  - else CodeS($\mathcal{O}(\mathcal{S})$)
- else
  - add $\mathcal{O}(\mathcal{S})$ to LIS

}

3. **Refinement Pass**

   For each entry $(i, j, k) \in$ LSP, except those included in the last sorting pass, output the $n^{th}$ MSB of $\mid c_{i,j,k} \mid$.

4. **Quantization Step**

   Decrement $n$ by 1 and go to step 2.

The initial sets $\mathcal{S}$ in the LIS are the subbands of the 3D wavelet decomposition. In the first pass at the highest $n$, 3D-SPECK tests the significance of sets $\mathcal{S}$ in the LIS following the order of lowpass bands to highpass bands. As an example, for one-level decomposition, the scanning order is LLL, LHL, HLL, LLH, HHL, HLH, LHH, HHH. For higher level decomposition, the scanning path starts from the top of the pyramid down to its bottom by following the same order from lowpass bands to highpass bands.

ProcessS and CodeS are extensions of the two-dimensional procedures ProcessS and CodeS of SPECK. 3D-SPECK processes a type $\mathcal{S}$ set by calling ProcessS to test its significance with respect to a threshold $n$. If not significant, it stays in the LIS. Otherwise, ProcessS will call CodeS to partition set $\mathcal{S}$ into eight approximately equal subsets $\mathcal{O}(\mathcal{S})$ (Fig 4). 3D-SPECK then treats each of these subsets as new type $\mathcal{S}$ set, and in turn, tests their significance. This process will be executed recursively until reaching pixel level where the significant pixel in the original set $\mathcal{S}$ is located. The algorithm then sends the significant pixel to the LSP, outputs a bit 1 to indicate the significance of the pixel, and outputs another bit to represent the sign of the pixel.

After finished the processing of all sets in the LIS, 3D-SPECK executes the refinement pass. The algorithm outputs the $n^{th}$ most significant bit of the absolute value of each entry $(i, j, k)$ in the LSP, except those included in the just-completed sorting pass. The procedure refines significant pixels that were found during previous passes progressively.

The last step of the algorithm is to decrease $n$ by 1 and return to the sorting pass of the current LIS, making the whole process run recursively.

The decoder is designed to have the same mechanism as the encoder. Based on the outcome of the significance tests in the received bit stream, the decoder can follow exactly the same execution path as the encoder and therefore reconstruct the image sequence progressively.

### 3.3 Processing Order of Sets

During the sorting pass, we claim that sets should be tested in increasing order of size. This follows the argument in [11, 12]. After the first pass, many sets of type $\mathcal{S}$ of varying sizes are generated and added to the LIS. For instance, the algorithm searches a set $\mathcal{S}$ and finds some significant pixels against the current threshold $n$ belonging to set $\mathcal{S}$. Neighboring pixels in set $\mathcal{S}$ not found to be significant in the current pass and sent to the LIS are very likely to be found as significant against the next lower threshold. Furthermore, our experiments show that a large number of sets with size of one are generated after the first iteration. Therefore, testing sets in increasing order of size can test pixel level first and locate new significant pixels immediately.

We do not use any sorting mechanism to process sets of type $\mathcal{S}$ in increasing order of their sizes. Even the fastest sorting algorithm will slow down the coding procedure significantly. This is not desirable in fast implementation of

coders. However, there are simple ways of completely avoiding this sorting procedure. SPECK uses an array of lists to avoid sorting, whereas we use a different and simpler approach to achieve this goal. 3D-SPECK only maintains one list instead of an array of lists.

Note that the way sets $\mathcal{S}$ are constructed, they lie completely within a subband. Thus, every set $\mathcal{S}$ is located at a particular level of the pyramidal structure. Each time the algorithm partitions a set $\mathcal{S}$, it generates eight smaller sets in approximate equal sizes, and the sizes of these sets $\mathcal{S}$ corresponds to a higher level of the pyramid. Based on this fact, to process sets of type $\mathcal{S}$ in increasing order of their sizes, we only need to search the same LIS several times at each iteration. Each time we only test sets with sizes corresponding to a particular level of the pyramid. This implementation completely eliminates the need for any sorting mechanism for processing the sets $\mathcal{S}$. Thus, we can test sets in increasing order of size while keeping our algorithm running fast.

### 3.4 Entropy Coding

As with other coding methods such as SPIHT and SPECK, the efficiency of our algorithm can be improved by entropy-coding its output [16][20][24][30]. We use the adaptive arithmetic coding algorithm of Witten *et al.* [32] to code the significance map. Referring to the function CodeS in our algorithm, instead of coding the significance test results of the eight subsets separately, we code them together first before further processing the subsets. Simple context is applied for conditional coding of the significance test result of this subset group. More specifically, we encode the significance test result of the first subset without any context, but encode the significance test result of the second subset by using the context (whether the first subset is significant or not) of the first coded subset and so on. Other outputs from the encoder are entropy coded by applying simple arithmetic coding without any context.

Also, we make the same argument as SPECK that if a set $\mathcal{S}$ is significant and its first seven subsets are insignificant, then this means that the last subset must be significant. Therefore, we do not need to test the significance of the last subset. This reduces the bit rate somewhat and provides corresponding gains.

We have chosen here not to entropy-code the sign and magnitude refinement bits, as small coding gains are achieved only with substantial increase in complexity. The SPECK variant, EZBC [9], has chosen this route, along with

a more complicated context for the significance map quadtree coding. The application will dictate whether the increase in coding performance is worth the added complexity.

### 3.5 Computational Complexity and Memory

3D-SPECK has low computational complexity. The algorithm is very simple, consisting mainly of comparisons, and does not require any complex computation. Since the analysis in detail has already been done in [11] and [12], we do not repeat it here. Readers are referred to these two papers for more information.

3D-SPECK also has low dynamic memory requirements. At any given time during the coding process, only one connected prism is processed.

### 3.6 Scaling Wavelet Coefficients by Bit Shifts

The integer filter transform with dyadic decomposition structure is not unitary. This does not affect the performance of lossless compression. However, to achieve good lossy coding performance, it is important to have an unitary transform. If the transform is not unitary, the quantization error in the wavelet domain is, thus, not equal to the mean squared error (MSE) in the time domain. Therefore, the lossy coding performance will be compromised. Appropriate transform structure and scaling the integer wavelet coefficients can make the transform approximately unitary before quantization. It is therefore possible to keep track of the final quantizer coding error with the integer transform.

We adopt the transform structure mentioned in [34]. As shown in Figure 5, a 4-level 1D wavelet packet tree structure is applied on the spectral axis. The scaling factors for each subband is indicated in the figure. As each scaling factor is some power of two, we can implement the scaling factor by bit shifting.

For the spatial axes, we keep the same 2D dyadic wavelet transform to each slice. As shown in Figure 6, 4-level dyadic decomposition structure with scaling factor for each subband is plotted. Each of the scaling factors is some power of two and therefore can be implemented by bit shifting.

To summarize, the 3D integer wavelet packet transform we use here is first to apply 1D packet decomposition and bit shifting along the spectral axis, followed by the basic 2D dyadic decomposition and bit shifting on the spatial axes. An example is shown in Figure 7, where scaling factors associated with

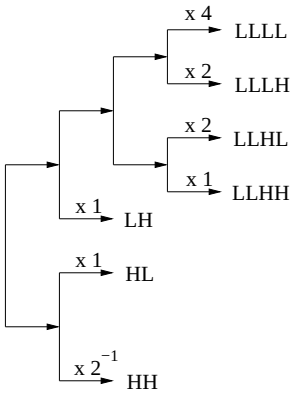


Figure 5: 1D wavelet packet structure along the spectral axis that makes the transform approximately unitary by shifting of the integer wavelet coefficients.

some subbands are indicated. The factors are the multiplications of the corresponding scaling factors in Fig. 5 and Fig. 6. The actual implementation of scaling factors is to shift all coefficients to some positive numbers of power two. In other words, for our case, all factors used in the implementation are four times of the factors shown in Fig. 7. This 3D integer wavelet packet structure makes the transform approximately unitary, and thus leads to much better lossy coding performance.

### 3.7 Resolution Progressive 3D-SPECK

Although the subband transform structure is inherently scalable of resolution, most embedded coders in the literature are unable to efficiently provide resolution scalable code streams. This is a consequence of entanglement in coding, modeling, and data structure across different resolutions. As an example, EZW and SPIHT, the classical zerotree coders with individual zerotrees spanning several subband scales are not efficient for resolution scalable coding.

However, we can modify our implementation of 3D-SPECK quite easily to enable resolution scalability. The idea is just to run the 3D-SPECK algorithm on each subband separately. Instead of maintaining the same significance threshold across subbands until we exhaust them, we maintain separate LIS and

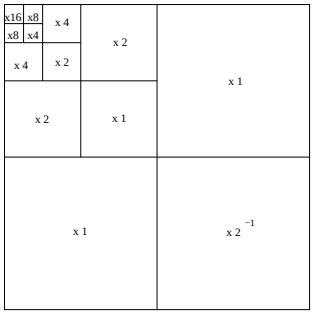


Figure 6: Same 2D spatial dyadic wavelet transform for each slice.

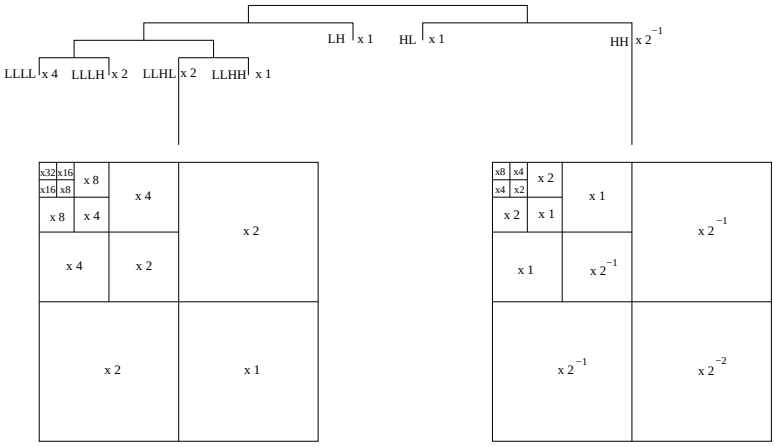


Figure 7: 3D integer wavelet packet transform.

LSP lists for each subband and proceed through the lower thresholds in every subband before moving to the next one. Therefore, we generate an array of lists, $\{\mathrm{LSP_k}\}$ and $\{\mathrm{LIS_k}\}$, where $k$ is the subband index. We move through the subbands in the same order as before, from lowest to highest scale. Therefore, we can truncate the bit stream corresponding to a reduced scale and decode to that scale.

The generation of the bit stream successively by subbands in order of increasing scale is illustrated in Figure 8 for a two level dyadic decomposition into $K = 15$ subbands. The code stream for each subband is SNR scalable, and the full bit stream is resolution scalable, because it can be terminated at any

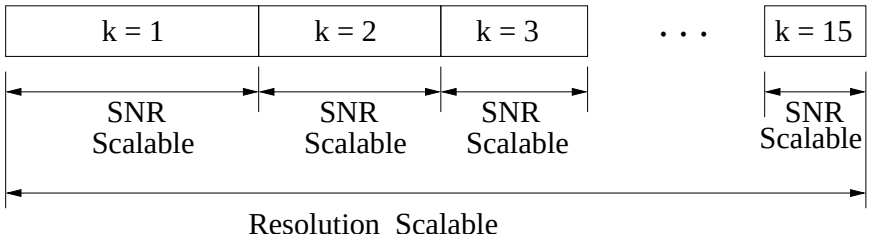


Figure 8: Bitstream strucure of resolution progressive 3D-SPECK

subband corresponding to lower than full scale and be decoded to that scale.

In order to effect multi-resolution decoding, bit stream boundaries are maintained between subbands. Adaptive arithmetic coding models are accumulated from samples within the same resolution scale. Finally, the modeling contexts do not include any neighbor from the finer scales. These conditions guarantee the decodability of the truncated code stream.

Although the full bit stream is resolution progressive, it is now not SNR progressive. The bit streams in the subbands are individually SNR progressive, so the bits belonging to the same bit planes in different subbands could be interleaved at the decoder after truncation to the desired scale to produce an SNR scalable bit stream.

For the SNR progressive coding mode, the bits are allocated optimally across subbands, according to the significance threshold of the coefficients. But, for the resolution progressive mode, for a given target bit rate, we need now to apply an explicit bit allocation algorithm to assign different bit rates to different subbands to minimize mean squared error. The solution is the same rate-distortion slope for every subband receiving non-zero rate. That slope depends on the target rate. JPEG2000 uses over-coding to some high rate in every code block and calculates these rate-distortion slopes while encoding. Then the bit stream in each code block is truncated to the correct point in an iterative procedure.

For the sake of expediency, we adopt a different procedure [37, 36] to solve the rate allocation. Let us assume that the subbands are numbered acording to decreasing variance to size ratio $\sigma_k^2/n_k$, $k = 1, 2, \ldots, K$. At low rates, the subbands numbered from some subband, say $m$, to $K$, will receive zero rate.

Thus, if the first $m$ subbands obtain a nonzero rate, the optimum rate allocation is

$$r_k = \begin{cases} \frac{N}{N_{s,m}} R + \frac{1}{2} \log_2 \frac{\sigma_k^2/n_k}{\gamma_m^2}, & 1 \le k \le m \\ 0, & m < k \le K \end{cases} \tag{3}$$

where $\gamma_m^2$ is a size weighted geometric mean of the variances of the first $m$ subbands and is defined as

$$\gamma_m^2 = [\prod_{k=1}^{m} (\sigma_k^2/n_k)^{n_k}]^{1/N_{s,m}}$$

where $n_k$ is the number coefficients in subband $k$ and $N_{s,m} = \sum_{k=1}^{m} n_k$ is the number of coefficients in the first $m$ subbands.

In order to find $m$, we use the following iterative procedure.

1. Set $m = K$.
2. Allocate rate using Equation 3.
3. If all $m$ bands are allocated non-negative rates, the rate allocation is complete. Otherwise, decrement $m$ by 1 and go to step 2.

To decode the image sequence to a particular level at a given rate, we need to encode each subband at a higher rate so that the algorithm can truncate the sub-bitstream to the assigned rate. However, unlike the JPEG2000 method, with this method we can achieve the target rate at the encoding stage without over-coding.

## 4 Numerical Results

We performed coding experiments on three signed 16-bit reflectance AVIRIS image volumes. AVIRIS has 224 bands and 614 $\times$ 512 pixel resolution that corresponds to an area of approximately 11 km $\times$ 10 km on the ground. We have 1997 runs of Moffett Field scene 1 and 3 and Jasper Ridge scene 1. For our experiments, we cropped each scene to 512 $\times$ 512 $\times$ 224 pixels.

| File Name | Coding Methods | | | | |
|---|---|---|---|---|---|
| | 3D-SPECK | 3D-SPIHT | JP2K-Multi | 2D-SPIHT | JPEG 2000 |
| moffett scene 1 | 6.9102 | 6.9411 | 7.1748 | 7.9714 | 8.7905 |
| moffett scene 3 | 6.8209 | 6.7402 | 7.0021 | 7.5847 | 7.7258 |
| jasper scene 1 | 6.7014 | 6.7157 | 6.8965 | 7.9770 | 8.5860 |

Table 1: Comparison of methods for Lossless coding of test 16 bit image volumes. The data are given in bits per pixel per band (bpppb), averaged over the entire image volume.

### 4.1 Comparison of Lossless Compression Performance

Table 1 presents the lossless performances of 3D-SPECK, 3D-SPIHT[3], JPEG-2000 multi-component (JP2K-Multi) [13], 2D-SPIHT and JPEG2000 [13]. JP2K-Multi is implemented first by applying the S+P filter on the spectral (wavelength) dimension and is then followed by application of the two-dimensional JPEG2000 on the spatial dimensions. S+P integer filters are used for 3D-SPECK, 3D-SPIHT and 2D-SPIHT, while for JPEG2000, the integer filter (5,3) [4] is used. For all 3D algorithms, including 3D-SPECK, 3D-SPIHT and JP2K-Multi, the results of AVIRIS data are obtained by coding all 224 bands as a single unit, and for the two 2D algorithms, the results are obtained by first coding the AVIRIS data band by band and then averaging over the entire volume.

Overall, 3D algorithms perform better than 2D algorithms. Compared with 2D-SPIHT and JPEG2000, 3D-SPECK yields, on average, 13.1% and 18.6% decreases in compressed file sizes for AVIRIS test image volumes. 3D-SPECK and 3D-SPIHT are fairly comparable as their results are quite close. They both out perform the benchmark JPEG2000 multi-component, averaged over the three image volumes, by 3.0% and 3.2% decreases in file size, respectively. Surprisingly, considering its considerably higher complexity, JPEG2000 is not as efficient as 2D-SPIHT. As shown in the table, 2D-SPIHT always yields smaller bits per pixel per band (bpppb) than that of JPEG2000.

[3]We use symmetric tree 3D-SPIHT here.

### 4.2 Comparison of Lossy Compression Performance

As we stated above, many hyperspectral image applications can perform reliably on images compressed to very low bit rates. Therefore, we present lossy versions of our algorithm as well by comparing the performances with other lossy algorithms.

To quantify fidelity, the coding performances are reported using rate-distortion results, by means of Signal-to-Noise ratio for the whole sequence (SNR):

$$\mathrm{SNR} = 10 \log_{10} \frac{P_x}{\mathrm{MSE}} \ \mathrm{dB} \tag{4}$$

where $P_x$ is the average squared value (power) of the original AVIRIS sequence, and MSE is the mean squared error over the entire sequence.

We present the lossy compression performance of 3D-SPECK by using both integer filter implementation, which enables lossy-to-lossless compression, and floating point filter implementation, which provides better lossy performance. For both implementations, the coding results are compared with 3D-SPIHT and JP2K-Multi.

The rate-distortion results for 3D-SPECK, 3D-SPIHT, and JP2K-Multi integer implementations are listed in Table 2 for our three test image volumes. For each algorithm with a desired bit rate, we truncate the same embedded information sequence obtained in section 4.1 at appropriate points and reconstruct the data to the corresponding accuracy. If no information loss is allowed, the whole embedded sequence will be used to fully recover the original image volume.

Overall, both 3D-SPECK and 3D-SPIHT perform better than JPEG2000 multi-component, providing higher SNR all the time. For all three test image volumes, the results show that 3D-SPECK is comparable to 3D-SPIHT, being slightly worse for moffett scene 3, but slightly better for moffett scene 1 and jasper scene 1.

Table 3 shows the rate-distortion results of the lossy implementations for 3D-SPECK, 3D-SPIHT and JPEG2000 multi-component. All results are obtained by using all 224 bands as a single coding unit and 5-level pyramid decompositions with the 9/7 tap biorthogonal filters and using a reflection extension at the image edges. Since every codec is embedded, the results for various bit rates can be obtained from a single encoded file. Comparing with integer filter results at different reconstruction rates, 3D-SPECK and 3D-SPIHT floating point implementations have better performances, both yielding approximate a

| Method | SNR (dB) vs. Rate (bpppb) | | | | | |
|---|---|---|---|---|---|---|
| | **0.1** | **0.2** | **0.5** | **1.0** | **2.0** | **4.0** |
| moffett scene 1 | | | | | | |
| 3D-SPECK | 15.717 | 20.778 | 29.199 | 37.284 | 44.731 | 54.605 |
| 3D-SPIHT | 15.509 | 20.605 | 29.105 | 37.198 | 44.671 | 54.544 |
| JP2K-Multi | 14.770 | 19.655 | 27.999 | 36.312 | 44.460 | 53.686 |
| moffett scene 3 | | | | | | |
| 3D-SPECK | 10.622 | 16.557 | 25.998 | 34.845 | 42.002 | 49.721 |
| 3D-SPIHT | 10.828 | 16.740 | 26.102 | 34.946 | 42.094 | 49.892 |
| JP2K-Multi | 10.264 | 15.952 | 25.208 | 33.835 | 41.535 | 49.240 |
| jasper scene 1 | | | | | | |
| 3D-SPECK | 19.022 | 22.675 | 30.400 | 36.697 | 43.622 | 51.857 |
| 3D-SPIHT | 18.905 | 22.553 | 30.279 | 36.647 | 43.566 | 51.742 |
| JP2K-Multi | 17.825 | 21.869 | 29.035 | 36.039 | 42.516 | 51.124 |

Table 2: Comparative evaluation the rate distortions of integer filter versions of 3D-SPECK, 3D-SPIHT and JPEG2000 multi-component.

range of 1.5 to 3.5 dB higher SNR. For JPEG2000 multi-component, the gains over the integer filter implementation are smaller. 3D-SPECK and 3D-SPIHT are competitive as they demonstrate quite close rate-distortions results for all AVIRIS sequences. 3D-SPECK performs slightly worse for moffett scene 3 but slightly better for moffett scene 1 and jasper scene 1. JPEG2000 is again worse than the 3D-SPIHT and 3D-SPECK in all trials.

These coding gains will produce better performance for hyperspectral applications. Therefore, for hyperspectral applications such as classification and feature extraction that can function reliably with lossy representations, the floating pointer filter version is a better choice.

### 4.3 Resolution Progressive 3D-SPECK Results - Floating Point Filters

As before, the coding performances of resolution progressive 3D-SPECK are reported in signal-to-noise ratio (SNR), using mean square error (MSE) calcu-

| Method | SNR (dB) vs. Rate (bpppb) | | | | | |
|---|---|---|---|---|---|---|
| | **0.1** | **0.2** | **0.5** | **1.0** | **2.0** | **4.0** |
| moffett scene 1 | | | | | | |
| 3D-SPECK | 16.671 | 21.520 | 29.913 | 38.595 | 47.178 | 55.574 |
| 3D-SPIHT | 16.570 | 21.461 | 29.880 | 38.539 | 47.136 | 55.527 |
| JP2K-Multi | 15.286 | 19.920 | 28.194 | 36.558 | 45.430 | 55.177 |
| moffett scene 3 | | | | | | |
| 3D-SPECK | 12.604 | 17.983 | 26.988 | 35.370 | 44.095 | 50.786 |
| 3D-SPIHT | 12.924 | 18.249 | 27.277 | 35.620 | 44.371 | 51.037 |
| JP2K-Multi | 10.791 | 16.810 | 25.822 | 33.439 | 38.778 | 40.826 |
| jasper scene 1 | | | | | | |
| 3D-SPECK | 19.702 | 23.658 | 31.750 | 38.552 | 45.997 | 52.361 |
| 3D-SPIHT | 19.589 | 23.586 | 31.480 | 38.363 | 45.853 | 52.261 |
| JP2K-Multi | 18.246 | 22.172 | 29.813 | 36.625 | 43.369 | 51.963 |

Table 3: Comparative evaluation the rate distortions of floating point filter versions of 3D-SPECK, 3D-SPIHT and JPEG2000 multi-component.

lated over the whole sequence.

Figure 9 shows the reconstructed band 20 of jasper scene 1 decoded from a single scalable code stream at a variety of resolutions at 0.5 bpppb. The SNR values for 0.5 bpppb for the whole sequence are listed in Table 4 along with results of other bit rates. The corresponding bit budget for the individual resolutions are provided in Table 5. We can see that the computational cost of decoding reduces from one resolution level to the next lower one.

The SNR values listed in Table 4 for low resolution image sequences are calculated with respect to the reference image generated by the same analysis filter bank and synthesized to the same scale. The total bit cost decreases rapidly with a reduction in resolution. However, the image quality is increasingly degraded from one resolution to the next lower one. If we look at the sample images in Fig. 9, when the reconstructed sequences are presented at same display resolution, the perceived distortion for viewing a sample image at half resolution is equivalent to that at full resolution but from twice a distance. The low resolution sequences can thus allowed to be coded relatively coarsely.

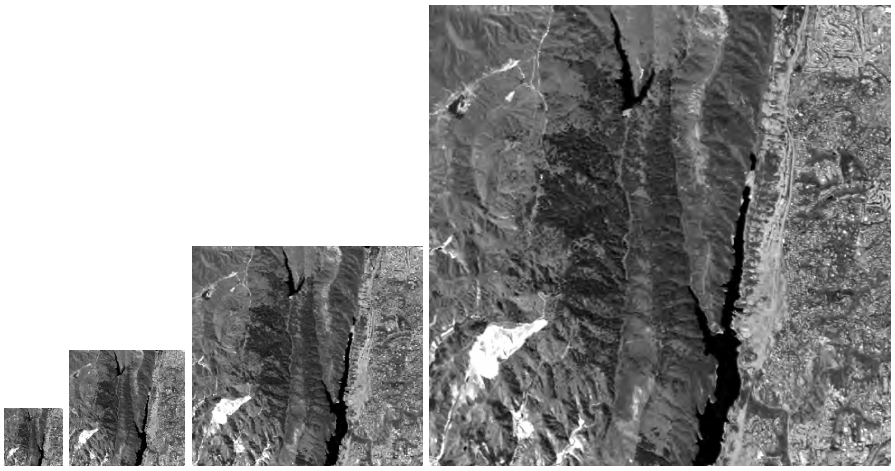

Figure 9: A visual example of resolution progressive 3D-SPECK using floating point wavelet transform. From left to right: 1/8 resolution, 1/4 resolution, 1/2 resolution, at 0.5 bpppb and full resolution (original).

The multi-resolution coding with such a perceptual concept has been reported in [9].

### 4.4 Resolution Progressive 3D-SPECK Results - Integer Filters

The basic function of integer filter implementation of resolution progressive is the same as the floating point implementation. The difference is that the former one supports lossy-to-lossless encoding/decoding, and thus lossy and lossless reconstructions can be generated from the same embedded bit stream.

Again we use the AVIRIS jasper scene 1 as an example. Coding the entire sequence as one unit, the GOF then is 224. To assess the visual quality of the reconstructed sequence, we show one band of the sequence. Figure 10 shows the reconstructed band 20 of jasper scene 1 losslessly decoded at a variety of resolutions.

For the lossy reconstruction from the same bit stream, Table 6 lists the SNR results of the whole jasper scene 1 sequence reconstructed to different resolution levels at different bit rates. The SNR values decrease from one resolution

| Bit Rate (bpppb) | SNR (dB) vs. scale | | | |
|---|---|---|---|---|
| | 1/8 | 1/4 | 1/2 | Full |
| 0.1 | 8.46 | 12.94 | 16.75 | 19.29 |
| 0.5 | 18.29 | 23.04 | 27.24 | 31.53 |
| 1.0 | 21.97 | 26.41 | 31.97 | 38.37 |
| 2.0 | 29.02 | 34.15 | 39.61 | 45.82 |

Table 4: SNR in dB for coding jasper scene 1 at a variety of resolution and coding bit rates using resolution progressive 3D-SPECK with floating point filters.

| Bit Rate (bpppb) | Bit budget (Kbits) vs. scale | | | |
|---|---|---|---|---|
| | 1/8 | 1/4 | 1/2 | Full |
| 0.1 | 11 | 91 | 734 | 5872 |
| 0.5 | 57 | 459 | 3670 | 29360 |
| 1.0 | 115 | 918 | 7340 | 58720 |
| 2.0 | 229 | 1835 | 14680 | 117441 |

Table 5: Corresponding bit budgets for Table 4

to the next lower one. If we look at the sample images, when the reconstructed sequences are presented at same display resolution, the perceived distortion for viewing a sample image at half resolution is equivalent to that at full resolution but from twice a distance. These results are similar to the floating point 3D-SPECK results. However, comparing to the values listed in Table 9, we could see that the floating point implementation performs better than the integer filter implementation on lossy image compression, demonstrating higher SNR values at the same bit rates.

Table 7 lists the related CPU times for Table 6 at the percentages of decoding time of lossless decoding at full resolution. We can see that the computational cost of decoding reduces from one resolution level to the next lower one.

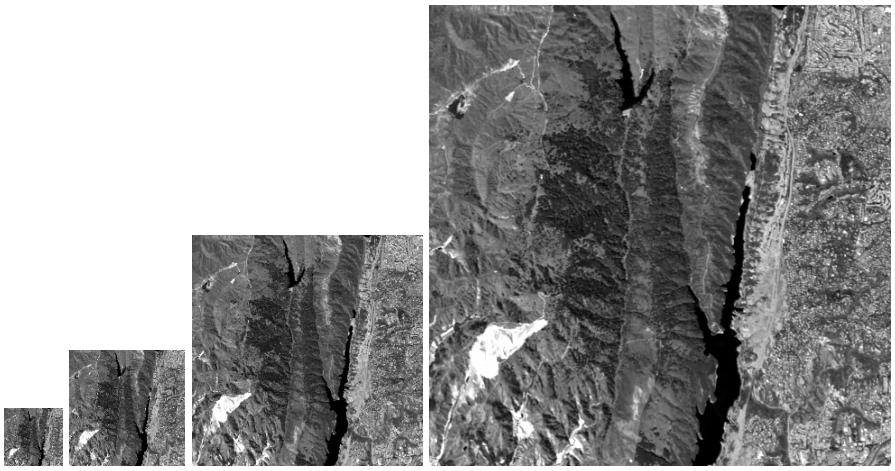

Figure 10: A visual example of resolution progressive 3D-SPECK with integer wavelet transform. From left to right: 1/8 resolution, 1/4 resolution, 1/2 resolution, and full resolution (original).

### 4.5 The Coding Gain

We wish to establish the effectiveness of extension of SPECK from 2D to 3D. We use jasper scene 1 as an example to show the lossy compression performance of 2D-SPECK and 3D-SPECK.

Table 8 shows the rate distortions results by comparing the results of 3D-SPECK and 2D-SPECK. The gap between 3D-SPECK and 2D-SPECK has the approximate range of 12 to 26 dB SNR, all depending on different bit rates. This demonstrates the big benefit of using 3D algorithm. 3D algorithm exploits the correlations along the spectral axis and therefore achieves much better coding performance.

### 4.6 Spectral Profile Integrity and Classification Performance

As the most important information for hyperspectral users is the spectral profile, we illustrate the performance for the integer filter version of 3D-SPECK by plotting the original spectral profiles of individual pixels, along with associated reconstructed and error profiles. Figure 11 and figure 12 show

| Bit Rate (bpppb) | SNR (dB) vs. Scale | | | |
|---|---|---|---|---|
| | 1/8 | 1/4 | 1/2 | Full |
| 0.1 | 7.92 | 12.20 | 16.27 | 18.75 |
| 0.5 | 17.72 | 22.25 | 26.95 | 30.30 |
| 1.0 | 21.13 | 25.86 | 30.78 | 36.52 |
| 2.0 | 28.62 | 33.36 | 38.94 | 43.22 |

Table 6: SNR values in dB for coding jasper scene 1 at a variety of resolution and coding bit rates using resolution progressive 3D-SPECK with integer filters.

the profiles for one asphalt pixel and one vegetation pixel of Jasper scene 1, respectively. The spectral profiles are preserved excellently even at 1.0 bppp, with only several larger values of errors occur at the spectral valleys around bands 160 and 224. The largest error corresponds to 2.4% of the maximum value. Increasing the bit rate, the error (difference) values drop quickly. The absolute values of errors are already within 25 at 2.0 bppp, corresponding to 0.7% and 0.6% of the maximum values for asphalt pixel and vegetation pixel, respectively. For bit rate of 4.0 bppp, as shown in Figure 11 and Figure 12, the differences between the original profiles and the reconstructed ones are barely distinguishable, and the errors are very small.

To address how our compression algorithm impacts remote sensing applications, it is important to provide an experiment for an end-use application. We use a well-known remote sensing classification method, Spectral Angle Mapper (SAM) [3], to test the proposed algorithm 3D-SPECK as well as 3D-SPIHT and JPEG2000 multi-component (JP2K-Multi). SAM determines the similarity of the original and reconstructed spectrum by computing the normalized inner product between the two spectra. We assume first that the classification on the original image is correct, and then report the classification performance as percentage of "correctly" classified pixels. In other words, we report the percentage of pixels whose classification is the same by using both the original and reconstructed spectrum.

Table 9 lists the classification results for three classes (asphalt, vegetation and water) of Jasper scene 1. All algorithms tested here are integer filter implementations. We can see that the classification tasks investigated are robust with

| Bit Rate (bpppb) | Decoding time (%) vs. scale | | | |
|---|---|---|---|---|
| | 1/8 | 1/4 | 1/2 | Full |
| 0.1 | 0.23 | 1.59 | 3.17 | 3.44 |
| 0.5 | 0.25 | 1.64 | 13.22 | 13.52 |
| 1.0 | 0.28 | 1.69 | 13.23 | 34.77 |
| 2.0 | 0.33 | 1.84 | 13.28 | 70.44 |
| lossless | 0.43 | 2.19 | 13.52 | 100 |

Table 7: Decoding times in percentage (%) of lossless decoding at full resolution for coding jasper scene 1 at a variety of resolution using resolution progressive 3D-SPECK.

| Method | SNR (dB) vs. Rate (bpppb) | | | | | |
|---|---|---|---|---|---|---|
| | 0.1 | 0.2 | 0.5 | 1.0 | 2.0 | 4.0 |
| 3D-SPECK | 19.702 | 23.658 | 31.750 | 38.552 | 45.997 | 52.361 |
| 2D-SPECK | 7.503 | 9.237 | 11.075 | 13.824 | 18.427 | 28.525 |

Table 8: Comparative evaluation the rate distortions of floating point filter versions of 3D-SPECK and 2D-SPECK.

respect to lossy compression of the source image. The percentage of correctly classified pixels converges to 100% at the rates higher than 1 bppp for all three algorithms, with JPEG2000 multi-component being slightly worse than that of 3D-SPECK and 3D-SPIHT. For 3D-SPECK and 3D-SPIHT, the distortions in the reconstructed data caused by the compression process result in only minor losses in classification accuracy even at low bit rate such as 1 bppp, with the classification accuracy higher than 99% almost all the time. For 3D-SPECK and 3D-SPIHT at very low bit rate such as 0.2 bppp, the percentages of classification accuracy are already higher than 97%. JPEG2000 multi-component provides much worse classification performances at 0.2 bppp. Overall, JPEG2000 multi-component performs not as well as the other two algorithms, rendering much poorer classification accuracy at very low bit rates and slightly lower percentage of classification accuracy at higher bit rate. Therefore, massively large

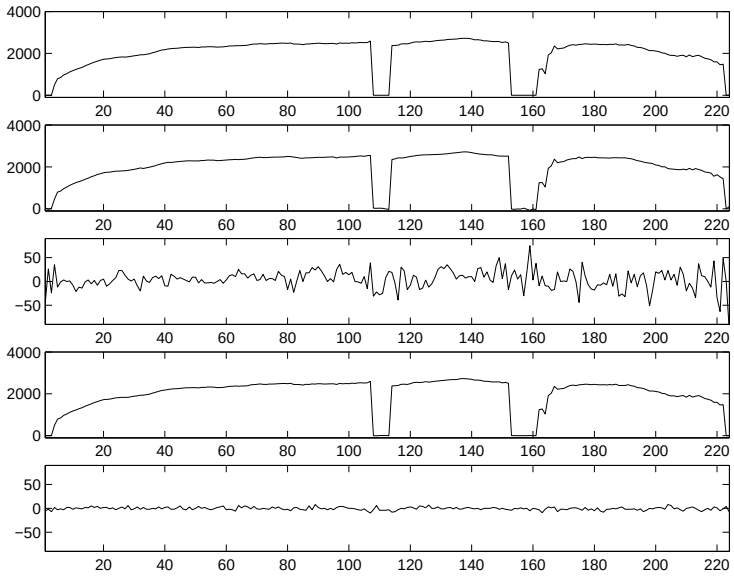


Figure 11: The original, reconstructed and the difference values between the original and reconstructed pixels for an asphalt pixel for Jasper scene 1. The first graph is the original, the second one is the reconstructed pixel at 1.0 bppp, the third one is the difference values at 1.0 bppp, the fourth one is the reconstructed pixel at 4.0 bppp, and the last one is the difference values at 4.0 bppp.

image data sets can be reduced to manageable sizes with only minor reductions in classifier performance.

# 5 Conclusion

This paper proposed a wavelet-based three-dimensional set partitioning embedded block coder for hyperspectral image compression. The three dimensional wavelet transform automatically exploits inter-band dependence and the set-partitioning algorithm, 3D-SPECK, efficiently codes the wavelet coefficients. Two versions of the algorithm were implemented. The integer filter implementation enables lossy-to-lossless compression, and the floating point filter implementation provides better performance for lossy representation. Wavelet packet structure and bit shifting were applied on the integer filter implementa-

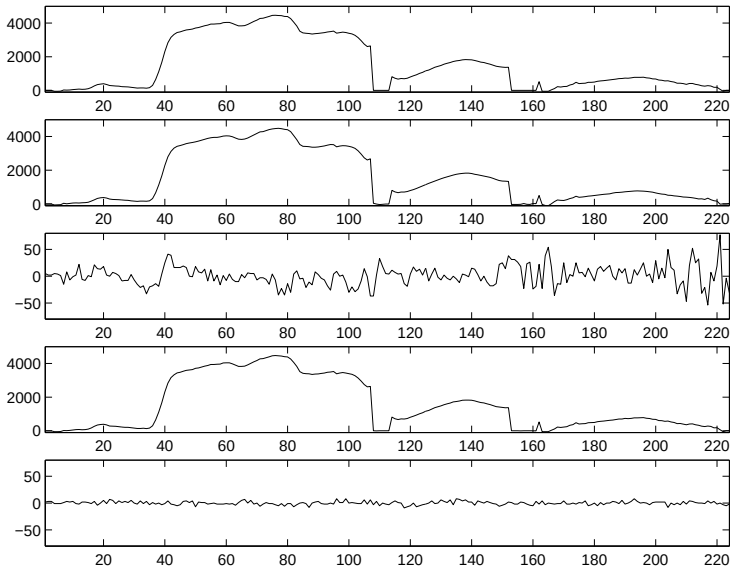


Figure 12: The original, reconstructed and the difference values between the original and reconstructed pixels for a vegetation pixel for Jasper scene 1. The first graph is the original, the second one is the reconstructed pixel at 1.0 bppp, the third one is the difference values at 1.0 bppp, the fourth one is the reconstructed pixel at 4.0 bppp, and the last one is the difference values at 4.0 bppp.

tion to make the transform approximately unitary.

Two versions of 3D-SPECK were presented, one with a rate-embedded bitstream giving inherent SNR scalability and the other with rate-embedded subband bitstreams giving inherent resolution scalability. The bitstream of the latter version could be re-organized at the decoder to provide an SNR-scalable full bitstream for the desired scale or resolution. This capability was not enabled in our decoder implementation, but could be in actual applications, if so desired.

Rate distortion results of both lossless and lossy compression of hyperspectral imagery have been presented, and all results were compared with other state-of-the-art three dimensional compression algorithms such as 3D-SPIHT and JPEG2000 multi-component. 3D-SPECK is competitive to 3D-SPIHT and better than JPEG2000 in compression efficiency. The plots of original, reconstructed and error spectral profiles shown that the proposed algorithm preserved

| | Classification Accuracy (%) vs. Rate (bpppb) | | | | | | |
|---|---|---|---|---|---|---|---|
| | **Methods** | **0.1** | **0.2** | **0.5** | **1.0** | **2.0** | **4.0** |
| Asphalt | 3D-SPECK | 89.22 | 97.91 | 98.51 | 99.87 | 99.97 | 99.98 |
| | 3D-SPIHT | 87.20 | 97.56 | 98.27 | 99.42 | 99.97 | 99.98 |
| | JP2K-Multi | 61.47 | 75.57 | 94.21 | 99.31 | 99.88 | 99.96 |
| Vege-tation | 3D-SPECK | 75.77 | 97.20 | 99.07 | 99.64 | 99.82 | 99.99 |
| | 3D-SPIHT | 80.35 | 97.83 | 99.57 | 99.84 | 99.90 | 99.99 |
| | JP2K-Multi | 65.27 | 84.40 | 95.17 | 98.99 | 99.58 | 99.93 |
| Water | 3D-SPECK | 85.77 | 97.70 | 99.36 | 99.45 | 99.72 | 99.97 |
| | 3D-SPIHT | 84.88 | 96.94 | 99.11 | 99.38 | 99.69 | 99.95 |
| | JP2K-Multi | 64.20 | 72.42 | 98.31 | 99.23 | 99.74 | 99.85 |

Table 9: Jasper scene 1 SAM classification.

spectral profiles well. The classification experiment with the three algorithms show 3D-SPECK correctly classifying above 97 % at rates as small as 0.20 bpppb (compression ratio of 80), roughly comparable to 3D-SPIHT, but clearly superior to JP2K-Multi.

The proposed 3D-SPECK is completely embedded and can be used for transmission progressive either by quality or resolution. These features make the proposed coder a good candidate to compress (encode) hyperspectral images for streaming applications, where the codestream can deliver to the user a certain sub-codestream that can be decoded according to the user's requirement of quality and resolution.

# References


[1] G.P. Abousleman, MW. Marcellin, and B.R. Hunt, *Hyperspectral image compression using entropy-constrained predictive trellis coded quantization,* IEEE Trans. Image Processing, Vol. 6, No. 4, April 1997.

[2] ISO/IEC 15444-2, Information Technology – JPEG 2000 Image Coding System – Part 2: Extensions, December 2000, Final Committee Draft.

[3] J.W. Boardman, F.A. Kruse and R.O. Green, *Mapping target signatures via partial unmixing of AVIRIS data*, Fifth JPL Airborne Earth Science Workshop, JPL Publication, pp.23-26, 1995.

[4] R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, *Wavelet transforms that map integers to integers*, J. Appl. Computa. Harmonics Anal. 5, pp.332-369, 1998.

[5] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, *Image coding using wavelet transform*, IEEE Trans. Image Processing, vol. 1, pp.205-220, 1992.

[6] P.L. Dragotti, G. Poggi, and A.R.P. Ragozini, *Compression of multispectral images by three-dimensional SPIHT algorithm*, IEEE Trans. on Geoscience and remote sensing, vol. 38, No. 1, Jan 2000.

[7] Thomas W. Fry, *Hyperspectral image compression on reconfigurable platforms*, Master Thesis, Electrical Engineering, University of Washington, 2001.

[8] J.C. Harsanyi, and C.I. Chang, *Hyperspectral image classification and dimensionality reduction: an orthogonal subspace projection approach*, IEEE Trans. Geoscience and Remote Sensing. Vol. 32, No. 4, July 1994.

[9] S-T. Hsiang and J.W. Woods, *Embedded image coding using zeroblocks of subband/wavelet coefficients and context modeling*, IEEE Int. Conf. on Circuits and Systems (ISCAS2000), vol. 3, pp.662-665, May 2000.

[10] P.F. Hsieh, *Classification of high dimensional data*, Ph.D. Thesis, Purdue University, 1998.

[11] A. Islam and W.A. Pearlman, *An embedded and efficient low-complexity hierarchical image coder*, in Proc. SPIE Visual Comm. and Image Processing, vol. 3653, pp. 294-305, 1999.

[12] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, *Efficient, Low-Complexity Image Coding with a Set-Partitioning Embedded Block Coder*, IEEE Trans. on Circuits and Systems for Video Technology, vol. 14, pp. 1219-1235, Nov. 2004.

[13] Kakadu JPEG2000 v3.4, http://www.kakadusoftware.com/.

[14] B. Kim and W.A. Pearlman, *An embedded wavelet video coder using three-dimensional set partitioning in hierarchical tree*, IEEE Data Compression Conference, pp.251-260, March 1997.

[15] Y. Kim and W.A. Pearlman, *Lossless volumetric medical image compression*, Ph.D Dissertation, Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, 2001.

[16] J. Li and S. Lei, *Rate-distortion optimized embedding*, in Proc. Picture Coding Symp., Berlin, Germany, pp.201-206, Sept. 10-12, 1997.

[17] S. Mallat, *Multifrequency channel decompositions of images and wavelet models*, IEEE Trans. Acoust., Speech, Signal Processing, vol. 37, pp.2091-2110, Dec. 1989.

[18] G. Motta, F. Rizzo, and J.A. Storer, *Compression of hyperspectral imagery*, Data Compression Conference. Proceedings. DCC 2003, pp. 25-27, March 2003.

[19] A.N. Netravali and B.G. Haskell, *Digital pictures, representation and compression*, in Image Processing, Proc. of Data Compression Conference, pp.252-260, 1997.

[20] E. Ordentlich, M. Weinberger, and G. Seroussi, *A low-complexity modeling approach for embedded coding of wavelet coefficients*, in Proc. IEEE Data Compression Conf., Snowbird, UT, pp.408-417, Mar. 1998.

[21] M.D. Pal, C.M. Brislawn, and S.P. Brumby, *Feature extraction from hyperspectral images compressed using the JPEG-2000 standard*, IEEE Southwest Symposium on Image Analysis and Interpretation, 5, pp.168-172, April. 2002.

[22] M.R. Pickering and M.J. Ryan, *Efficient spatial-spectral compression of hyperspectral data*, IEEE Trans. Geoscience and Remote Sensing, Vol. 39, No. 7, July 2001.

[23] M.J. Ryan and J.F. Arnold, *The lossless compression of AVIRIS images by vector quantization*, IEEE Trans. Geoscience and Remote Sensing, Vol. 35, No. 3, May 1997.

[24] *Proposal of the arithmetic coder for JPEG2000*, ISO/IEC/JTC1/SC29/WG1 N762, Mar. 1998.

[25] A. Said and W.A. Pearlman, *An image multiresolution representation for lossless and lossy compression,* IEEE Trans. Image Process. 5, pp.1303-1310, 1996.

[26] A. Said and W.A. Pearlman, *A new, fast and efficient image codec based on set partitioning in hierarchical trees,* IEEE Trans. on Circuits and Systems for Video Technology 6, pp.243-250, June 1996.

[27] P. Schelkens, *Multi-dimensional wavelet coding algorithms and implementations,* Ph.D dissertation, Department of Electronics and Information Processing, Vrije Universiteit Brussel, Brussels, 2001.

[28] J.M. Shapiro, *Embedded image coding using zerotrees of wavelet coefficients,* IEEE Trans. Signal Processing, vol. 41, pp.3445-3462, Dec. 1993.

[29] P. Simard, D. Steinkraus, and H. Malvar, *On-line adaptation in image coding with a 2-D tarp filter,* in Proceedings of the IEEE Data Compression conference, J.A. Storer and M.Cohn, Eds., Snowbird, UT, pp. 23-32, April 2002.

[30] D. Taubman, *High performance scalable image compression with EBCOT,* IEEE Trans. on Image Processing, vol. 9, pp.1158-1170, July, 2000.

[31] Yonghui Wang, Justin T. Rucker, and James E. Fowler, *3D tarp coding for the compression of hyperspectral images,* Submitted to IEEE Trans. on Geoscience and Remote Sensing, July 2003.

[32] I.H. Witten, R.M. Neal, and J.G. Cleary, *Arithmetic coding for data compression,* Commun. ACM, vol. 30, pp.520-540, June 1987.

[33] Z. Xiong, X. Wu, D.Y. Tun, and W.A. Pearlman, *Progressive coding of medical volumetric data using three-dimensional integer wavelet packet transform,* Medical Technology Symposium, 1998. Proceedings. Pacific, PP.384 -387, 1998.

[34] Z Xiong, X. Wu, S. Cheng, and J. Hua, *Lossy-to-lossless compression of medical volumetric data using three-dimensional integer wavelet transforms,* IEEE Trans. on Medical Imaging, Vol. 22, No. 3, March 2003.

[35] J. Xu, Z. Xiong, S. Li, and Y. Zhang, *Three-dimensional embedded subband coding with optimized truncation (3-D ESCOT)*, J. Applied and Computational Harmonic Analysis: Special Issue on Wavelet Applications in Engineering. vol. 10, pp.290-315, May 2001.

[36] W.A. Pearlman, *Performance Bounds for Subband Coding,* Chapter 1 in *Subband Image Coding*, J. W. Woods and Ed., Kluwer Academic Publishers, 1991.

[37] M. Balakrishnan and W.A. Pearlman, *Hexagonal subband image coding with perceptual weighting,* Optical Engineering, Vol. 32, No. 7, pp.1430-1437, July, 1993.