An embedded and efficient low-complexity hierarchical image coder

Asad Islam and William A. Pearlman

Electrical, Computer and Systems Engineering Dept.
Rensselaer Polytechnic Institute, Troy, NY 12180, USA

ABSTRACT

We propose an embedded hierarchical image coding algorithm of low complexity. It exploits two fundamental characteristics of an image transform — the well defined hierarchical structure, and energy clustering in frequency and in space. The image coding algorithm developed here, apart from being embedded and of low complexity, is very efficient and is comparable to the best known low-complexity image coding schemes available today.

Keywords: image coding, hierarchical coding, block entropy coding, embedded coding

1. INTRODUCTION

Image coding utilizing scalar quantization on hierarchical structures of transformed images has been a very effective and computationally simple technique. Shapiro[1] was the first to introduce such a technique with his Embedded Zerotree Wavelet (EZW) algorithm. Different variants of this technique have appeared in the literature which provide an improvement over the initial work. Said & Pearlman[2,3] successively improved the EZW algorithm by extending this coding scheme, and succeeded in presenting a different implementation based on a set-partitioning sorting algorithm.[4] This new coding scheme, called the Set-Partitioning in Hierarchical Trees (SPIHT), provided an even better performance than the improved version of EZW.[3]

All of these scalar quantized schemes employ some kind of significance testing of sets or groups of pixels, in which the set is tested to determine whether the maximum magnitude in it is above a certain threshold. The results of these significance tests determine the path taken by the coder to code the source samples. These significance testing schemes are based on some very simple principles which allow them to exhibit excellent performance. Among these principles are the partial ordering of magnitude coefficients with a set-partitioning sorting algorithm, bit plane transmission in decreasing bit plane order, and exploitation of self-similarity across different scales of an image wavelet transform.

An interesting thing to note in these schemes is that all of them have relatively low computational complexity, considering the fact that their performance is comparable to the best-known image coding algorithms. This feature seems in conflict with the well-known tenets of information theory[5] that the computational complexity of a stationary source (i.e., source sample aggregates) increases as the coding efficiency of the source increases. These coding schemes seem to have provided a breathing space in the world of simultaneously increasing efficiency and computational complexity.

An important characteristic that this class of coders possesses is the property of progressive transmission and embeddedness. Progressive transmission refers to the transmission of information in decreasing order of its information content. In other words, the coefficients with the highest magnitudes are transmitted first. Since all of these coding schemes transmit bits in decreasing bit plane order, this ensures that the transmission is progressive. Such a transmission scheme makes it possible for the bitstream te be embedded, i.e., a single coded file can used to decode the image at various rates less than or equal to the coded rate, to give the best reconstruction possible with the particular coding scheme.

With these desirable features of excellent performance and low complexity, along with others such as embeddedness and progressive transmission, it is no surprise that these scalar quantized significance testing schemes have recently

Asad Islam: E-mail: islama2@rpi.edu
William A. Pearlman: E-mail: pearlman@ecse.rpi.edu

become very popular in the search for practical, fast and efficient image coders, and in fact, have become the basis for serious consideration for future image compression standards.

The algorithm that we propose can be said to belong to this class of scalar quantized significance testing schemes. It has its roots primarily in the ideas developed in the SPIHT, AGP[6] and SWEET[7] image coding algorithms. It is different from some of the above-mentioned schemes in that it does not use trees which span, and exploit the similarity, across different subbands; rather, it makes use of sets in the form of blocks. A more detailed discussion describing the position of this coding scheme with respect to these coders is given in Section 5. The main idea is to exploit the clustering of energy in frequency and space in hierarchical structures of transformed images.

We call this image coding scheme the *Set Partitioned Embedded bloCK coder* (SPECK), and refer to it with this name in the discussion that follows. We present the characteristic features of the SPECK coder in Section 2. Section 3 provides the terminology and setup used in the algorithm, followed by Section 4 which lists the actual algorithm and describes its working in detail along with a discussion of the types of partitioning schemes used. Section 5 provides the motivation behind SPECK and explains its position among the class of similar hierarchical coding schemes. Section 6 gives the numerical and visual results obtained with this coding scheme, followed by the concluding statements in Section 7.

2. FEATURES OF THE CODER

The SPECK image coding scheme has all the properties characteristic of scalar quantized significance testing schemes. In particular, it exhibits the following properties:

- It is completely embedded - a single coded bitstream can be used to decode the image at any rate less than or equal to the coded rate, to give the best reconstruction of the image possible with the particular coding scheme.
- It employs progressive transmission - source samples are coded in decreasing order of their information content.
- It has low computational complexity - the algorithm is very simple, consisting mainly of comparisons, and does not require any complex computation.
- It has low dynamic memory requirements - at any given time during the coding process, only one connected region (lying completely within a subband) is processed. Once this region is processed, the next region is then considered for processing.
- It has fast encoding/decoding - this is due to the low-complexity nature of the algorithm.
- It has efficient performance - its efficiency is comparable to the other low-complexity algorithms available today.
- It can be used for lossy and lossless compression - depending on the choice of the transform, the algorithm can be used for lossless or nearly lossless coding, apart from lossy coding.

3. OUTLINE OF THE CODING METHOD

In this section, we give a brief outline of the SPECK coding scheme and an explanation of the terminology used in it. Consider an image $\mathcal{X}$ which has been adequately transformed using an appropriate subband transformation (most commonly, the discrete wavelet transform). The transformed image is said to exhibit a hierarchical pyramidal structure defined by the levels of decomposition, with the topmost level being the root. The finest pixels lie at the bottom level of the pyramid while the coarsest pixels lie at the top (root) level. The image $\mathcal{X}$ is represented by an indexed set of transformed coefficients $\{c_{i,j}\}$, located at pixel position (i,j) in the transformed image.

Pixels are grouped together in sets which comprise of regions in the transformed image. Following the ideas of SPIHT, we say that a set $\mathcal{T}$ of pixels is *significant* with respect to n if

$$\max_{(i,j)\in\mathcal{T}} \{|c_{i,j}|\} \geq 2^n ,$$

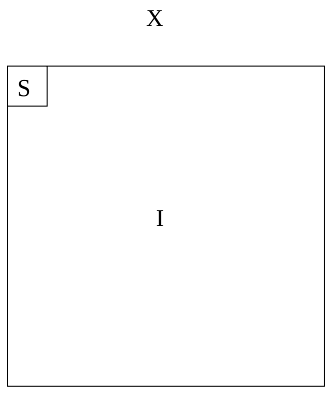

Figure 1. Partitioning of image $\mathcal{X}$ into sets $\mathcal{S}$ and $\mathcal{I}$

otherwise it is *insignificant*. We can write the significance of a set $\mathcal{T}$ as a function of n and the set $\mathcal{T}$, i.e.

$$\mathcal{S}_n(\mathcal{T}) = \begin{cases} 1, & \text{if } 2^n \le \max\limits_{(i,j)\,\in\,\mathcal{T}} |c_{i,j}| < 2^{n+1} \\ 0, & \text{else} \end{cases} \tag{1}$$

The SPECK algorithm makes use of rectangular regions of image. These regions or sets, henceforth referred to as sets of type $\mathcal{S}$, can be of varying dimensions. The dimension of a set $\mathcal{S}$ depends on the dimension of the original image and the subband level of the pyramidal structure at which the set lies.

We define the size of a set to be the cardinality $\mathcal{C}$ of the set, i.e., the number of elements (pixels) in the set.

$$\text{size}(\mathcal{S}) = \mathcal{C}(\mathcal{S}) \equiv |\mathcal{S}| \tag{2}$$

During the course of the algorithm, sets of various sizes will be formed, depending on the characteristics of pixels in the original set. Note that a set of size 1 consists of just one pixel.

The other type of sets used in the SPECK algorithm are referred to as sets of type $\mathcal{I}$. These sets are obtained by chopping off a small square region from the top left portion of a larger square region. A typical set $\mathcal{I}$ is illustrated in Fig. 1.

We maintain two linked lists: LIS - List of Insignificant Sets, and LSP - List of Significant Pixels. The former contains sets of type $\mathcal{S}$ of varying sizes which have not yet been found significant against a threshold n while the latter obviously contains those pixels which have tested significant against n. Alternatively, as will become obvious later on, we can use an array of smaller lists of type LIS, each containing sets of type $\mathcal{S}$ of a fixed size, instead of using a single large list having sets $\mathcal{S}$ of varying sizes. Use of multiple lists will speed up the encoding/decoding process.

4. THE SPECK ALGORITHM

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

The SPECK coding algorithm is illustrated in Fig. 2. The actual algorithm is given at the top (statements $1-4$) and consists of the initialization step, the sorting and refinement passes, and the quantization step. This is followed by a description of the functions used in the actual algorithm. These four functions are `ProcessS()`, `CodeS()`, `ProcessI()` and `CodeI()`.

```
1. Initialization
    • Partition image transform $\mathcal{X}$ into two sets: $\mathcal{S} \equiv$ root, and $\mathcal{I} \equiv \mathcal{X} - \mathcal{S}$ (see Fig. 1)
    • output $n = \lfloor \log_2(\max_{\forall(i,j)\in\mathcal{X}} |c_{i,j}|) \rfloor$
    • add $\mathcal{S}$ to LIS and set LSP $= \phi$
2. Sorting Pass
    • in increasing order of size $\mathcal{C}$ of sets
        – for each set $\mathcal{S} \in$ LIS,
            ∗ ProcessS($\mathcal{S}$)
    • ProcessI()
3. Refinement Pass
    • for each $(i,j) \in$ LSP, except those included in the last sorting pass, output the $n$th MSB of $|c_{i,j}|$
4. Quantization Step
    • decrement $n$ by 1, and go to step 2

ProcessS($\mathcal{S}$)
{
    • output $S_n(\mathcal{S})$
    • if $S_n(\mathcal{S}) = 1$
        – 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
    • else
        – if $\mathcal{S} \notin$ LIS, add $\mathcal{S}$ to LIS
}
CodeS($\mathcal{S}$)
{
    • Partition $\mathcal{S}$ into four equal subsets $\mathcal{O}(\mathcal{S})$ (see Fig. 3)
    • for each $\mathcal{O}(\mathcal{S})$
        – output $S_n(\mathcal{O}(\mathcal{S}))$
        – if $S_n(\mathcal{O}(\mathcal{S})) = 1$
            ∗ 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
}
ProcessI()
{
    • output $S_n(\mathcal{I})$
    • if $S_n(\mathcal{I}) = 1$
        – CodeI()
}
CodeI()
{
    • Partition $\mathcal{I}$ into four sets — three $\mathcal{S}$ and one $\mathcal{I}$ (see Fig. 4)
    • for each of the three sets $\mathcal{S}$
        – ProcessS($\mathcal{S}$)
    • ProcessI()
}
```

Figure 2. The SPECK Algorithm

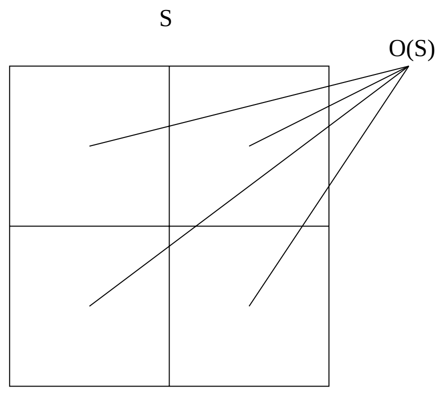

Figure 3. Partitioning of set $\mathcal{S}$

We will now describe the working of the SPECK algorithm presented in Fig. 2. We start with our source, a rectangular image $\mathcal{X}$, that has undergone an appropriate subband transformation. The image $\mathcal{X}$ consists of transformed coefficients $\{c_{i,j}\}$, located at pixel position (i, j). Such an image exhibits a hierarchical pyramidal structure having subbands at different levels of its decomposition. The topmost band is the root of the pyramid.

The algorithm starts by partitioning the image into two sets: set $\mathcal{S}$ which is the root of the pyramid, and set $\mathcal{I}$ which is everything that is left of the image after taking out the root (see Fig. 1). To start the algorithm, set $\mathcal{S}$ is added to LIS. We keep a note of the maximum threshold n_{max} such that $c_{i,j}$ is insignificant with respect to $n_{max}+1$, $\forall c_{i,j} \in \mathcal{X}$, but is significant against the threshold n_{max}, for some $c_{i,j} \in \mathcal{X}$.

4.1. Quadtree partitioning

Set $\mathcal{S}$ in LIS is processed by testing it for significance against the threshold $n = n_{max}$ (function `ProcessS`()). Assuming that $\mathcal{S}$ is significant, it is partitioned, by a quadtree partitioning process, into four subsets $\mathcal{O}(\mathcal{S})$, each having size approximately one-fourth the size of the parent set $\mathcal{S}$ (function `CodeS`()). Fig. 3 gives an illustration of this partitioning process.

Each of these four subsets are, in turn, treated as a set of type $\mathcal{S}$ and processed recursively until the pixel-level is reached where the pixels that are significant in the original set $\mathcal{S}$ are located and thereby coded. The pixels/sets that are found insignificant during this 'hunting' process are added to LIS to be tested against the next lower threshold later on. The motivation for quadtree partitioning of such sets is to zoom in quickly to areas of high energy in the set $\mathcal{S}$ and code them first.

4.2. Octave band partitioning

At this stage of the algorithm, there are no more sets of type $\mathcal{S}$ that need to be tested against n; if there were, they would be processed before going on to the next part of the algorithm. Once all sets of type $\mathcal{S}$ are processed, the set $\mathcal{I}$ is processed by testing it against the same threshold n (function `ProcessI`()). If it is found to be significant, it is partitioned by yet another partitioning scheme — the octave band partitioning. Fig. 4 gives an illustration of this partitioning scheme. Set $\mathcal{I}$ is partitioned into four sets — three sets of type $\mathcal{S}$ and one of type $\mathcal{I}$ (function `CodeI`()). The size of each of these three sets $\mathcal{S}$ is the same as that of the chopped portion of $\mathcal{X}$. The new set $\mathcal{I}$ that is formed by this partitioning process is now reduced in size.

The idea behind this partitioning scheme is to exploit the hierarchical pyramidal structure of the subband decomposition, where it is more likely that energy is concentrated at the top most levels of the pyramid and as one goes down the pyramid, the energy content decreases gradually. If a set $\mathcal{I}$ is significant against some threshold n, it is more likely that the pixels that cause $\mathcal{I}$ to be significant lie in the top left regions of $\mathcal{I}$. These regions are decomposed into sets of type $\mathcal{S}$, and are put next in line for processing.

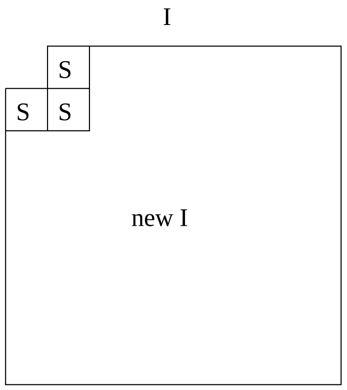

Figure 4. Partitioning of set $\mathcal{I}$

In this way, regions that are likely to contain significant pixels are grouped into relatively smaller sets and processed first, while regions that are likely to contain insignificant pixels are grouped into a large set. A single bit may be enough to code this large region against the particular threshold. Hence, once the set $\mathcal{I}$ is partitioned by the octave band partitioning method, the three sets $\mathcal{S}$ are processed in the regular image-scanning order, after which the newly formed reduced set $\mathcal{I}$ is processed.

It should be noted that processing the set $\mathcal{I}$ is a recursive process, and depending on the characteristics of the image, at some point in the algorithm, the set $\mathcal{I}$ will cover only the lower-most (bottom) level of the pyramidal structure. When, at this point, the set $\mathcal{I}$ tests significant against some threshold, it will be broken down into three sets $\mathcal{S}$ but there will be no new reduced set $\mathcal{I}$. To be precise, the new set $\mathcal{I}$ will be an empty set. Hence, the functions `ProcessI`() and `CodeI`() will have no meaning in the SPECK algorithm after this event.

Once all the sets $\mathcal{S}$ and $\mathcal{I}$ have been processed, the sorting pass for that particular threshold n is completed, after which the refinement pass is initiated which refines the quantization of the pixels in LSP, i.e. those pixels which were tested significant during the previous sorting passes. Once this is done, the threshold is lowered and the sequence of sorting and refinement passes is repeated against this lower threshold. This process is repeated until the desired rate is achieved or, in case of lossless or nearly lossless compression, all the thresholds up to the last, corresponding to $n = 0$, are tested.

4.3. Processing order of sets $\mathcal{S}$

An important step in the execution of the sorting pass comes after the first run of the algorithm. Once one pass has occurred, sets of type $\mathcal{S}$ of varying sizes are added to LIS. During the next lower threshold, these sets are processed in a particular order. The list LIS is not traversed sequentially for processing sets $\mathcal{S}$; rather, the sets are processed in increasing order of their size. In other words, say for a square image, sets of size 1 (i.e. pixels) are processed first, sets of size 4 (blocks of 2x2 pixels) are processed next, and so on.

The idea behind this strategy is that during the course of its execution, the algorithm sends those pixels to LIS whose immediate neighbors have tested significant against some threshold n but they themselves have not tested significant against that particular threshold. Chances are, because of energy clustering in the transform domain, that these insignificant pixels would have magnitudes close to the magnitudes of their neighboring pixels already tested significant, although lesser. So it is likely that these pixels will test positive to some nearby lower threshold and add to the reduction in the overall distortion of the coded image.

Moreover, the overhead involved in testing a single pixel and moving it to LSP is much lower than that involved in testing a group of pixels and moving the significant ones in it to LSP. Of course, if a whole sorting pass is completed, this scheme offers no advantage since all the sets of type $\mathcal{S}$ in LIS would be tested in either case. However, if the coding algorithm were to stop in the middle of a sorting pass, as it might if the desired rate is achieved, and the sets in LIS are processed in increasing order of their size, then we certainly get performance improvement.

It may seem that processing sets of type $\mathcal{S}$ in increasing order of their size involves a sorting mechanism — something which is not desirable in fast implementation of coders. However, there is a simple way of completely avoiding this sorting procedure.

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. Partioning a set $\mathcal{S}$ into four offsprings $\mathcal{O}(\mathcal{S})$ (i.e., forming sets $\mathcal{S}$ of a new reduced size) is equivalent to going down the pyramid one level at the corresponding finer resolution. Hence, the size of a set $\mathcal{S}$ for an arbitrary image corresponds to a particular level of the pyramid. If we use an array of lists, each corresponding to a level of the pyramid, then each list stores sets of a fixed size. Processing the lists in an order which corresponds to increasing size of sets completely eliminates the need for any sorting mechanism for processing the sets $\mathcal{S}$. Thus, we do not need to compromise the speed of the algorithm by employing some kind of sorting mechanism.

It should be noted that using an array of lists does not increase the memory requirements for the coder, as opposed to using a single list. This is because the total number of sets $\mathcal{S}$ that are formed during the coding process remain the same. Instead of storing these sets in one large list, we are storing them in several smaller lists with the aim of speeding up the coding process.

The decoder uses the same mechanism as the encoder. It receives significance test results from the coded bitstream and builds up the same list structure during the execution of the algorithm. Hence, it is able to follow the same execution paths for the significance tests of the different sets, and reconstructs the image progressively as the algorithm proceeds.

4.4. Entropy coding

Entropy coding of significance map is done using arithmetic coding with simple context-based models. Referring to the coding algorithm of Sec. 4, in the function `CodeS`(), the significance test results of the four subsets $\mathcal{O}(\mathcal{S})$ of set $\mathcal{S}$ (see Fig. 3) are not coded separately - rather, they are all coded together first before further processing the subsets. We use conditional coding for coding the significance test result of this 4-subset group. In other words, the significance test result of the first subset is coded without any context, while the significance test result of the second subset is coded using the context of the first coded subset, and so on. In this way, previously coded subsets form the context for the subset being currently coded.

Also, we make use of the fact that if a set $\mathcal{S}$ is significant and its first three subsets are insignificant, then this ensures that the fourth subset is significant and we do not have to send the significance test result of the last subset. This fact is utilized in reducing the bit budget. Results have shown that because of the nature of energy clustering in the pyramidal structure, the number of scenarios of the above mentioned type occur slightly more if the 4-group subsets are coded in reverse-scanning order than in the usual forward-scanning order. This saves some overhead in bit budget and provides corresponding gains.

5. DISCUSSION

The SPECK algorithm is motivated by the features inherent in the SPIHT, SWEET and AGP algorithms, and although it uses ideas from all these coding schemes, it is different from these coders in various respects.

The SPIHT coding scheme works by grouping pixels together in the form of spatial orientation trees. It is well known that a subband pyramid exhibits similarities across its subbands at the same spatial orientation. This property can be seen to be well illustrated if we partition the image transform in the form of spatial orientation trees. The SPIHT algorithm exploits this characteristic of the image transform by grouping pixels together into such structures.

The AGP algorithm, on the other hand, is a block-based coding algorithm and partitions the image transform in the form of blocks. The blocks are recursively and adaptively partitioned such that high energy areas are grouped together into small sets whereas low energy areas are grouped together in large sets. Such a type of adaptive quadtree partitioning results in efficient coding of the source samples. The SWEET coding algorithm is also block based and uses octave-band partitioning to exploit the pyramidal structure of image transforms.

Whereas SPIHT is a tree-based fully embedded coder which employs progressive transmission by coding bit planes in decreasing order, the AGP and SWEET coding algorithms are block-based coders which are *not* embedded and do not employ progressive transmission. SWEET codes a block upto a certain bit-depth, n_{min}, before moving on to the next one. Different rates of compressed images are obtained by appropriately choosing the minimum bit-plane, n_{min}, to which to encode. Finer set of compression ratios are obtained by scaling the image transform by some factor prior to the coefficient coding. The AGP algorithm is a block entropy coder which first reduces the alphabet of the image transform to a tractable level. It then partitions the image transform in the form of blocks and codes those blocks using a powerful entropy coding method.

Block-based coding is an efficient technique for exploiting the clustering of energy found in image transforms. It is a known fact that the statistics of an image transform vary remarkably as one moves from one spatial region to another. By grouping transform source samples in the form of blocks and coding those blocks independently, one is able to exploit the statistics of each block in an appropriate manner. This is one of the reasons that block-based coders work quite well. However, there is an increasing demand for some desirable properties for image coders, such as embeddedness and progressive transmission, which are very useful and much needed in the fast growing multimedia and networking environment. Both SWEET and AGP, although very efficient, do not possess these desirable properties.

The SPECK coding algorithm solves this problem by exhibiting these important properties lacking in most, if not all, block-based coding schemes. It is a fully embedded block-based coder which employs progressive transmission by coding bit planes in decreasing order. It employs octave-band partitioning of SWEET to exploit the hierarchical structure of the subband pyramid and concentrate more on potentially high energy subbands. It makes use of the adaptive quadtree splitting scheme of AGP to zoom into high energy areas within a region to code them with minimum significance maps. And it uses the significance map schemes of EZW and SPIHT to code the image transform progressively in decreasing bit-plane order. All this makes SPECK a very efficient block-based embedded image coding scheme.

6. NUMERICAL RESULTS

The following results were obtained with monochrome, 8 bpp, 512 x 512 images. We used 5-level pyramids constructed with the 9/7 tap biorthogonal filters[8] and using a reflection extension at the image edges. The bit rates are calculated from the actual size of the compressed files. Since the codec is embedded, the results for various bit rates are obtained from a single encoded file.

Tables 1 - 3 show the PSNR obtained by this coding method at the rates 0.25, 0.5 and 1.0 bpp for the three images 'lena', 'barbara' and 'goldhill'. These results are obtained by entropy-coding the bits put out by the coding scheme. A similar result is also included for the EZW, SPIHT and AGP coding schemes, which also employ entropy coding of the significance map.

The rate-distortion curves are also plotted in Fig. 5 for the three images at rates upto 1 bpp. The results are of the same nature as mentioned above for the three coding schemes. Figs. 6-9 show the Barbara (512x512) and Goldhill (512x512) images coded at the rates 0.25, 0.5 and 1.0 bpp using the SPECK coder. As we can see from the numerical as well as visual results, the SPECK coding scheme provides excellent results, comparable to the popular image coding schemes, such as SPIHT. For some images, such as Barbara, it gives slightly better performance than SPIHT.

7. CONCLUSIONS

We introduced a new block-based hierarchical image coding scheme, the SPECK image coder. The proposed algorithm is completely embedded, employs progressive transmission and is of low computational complexity. The algorithm is simple, thus providing fast encoding/decoding. The results are comparable to the best known low-complexity image coders available today. Moreover, this block-based coding scheme has the potential for small memory usage and work is being done in this regard.

The key to the algorithm is to properly and efficiently process different regions of the transformed image based on their energy content. The proposed coder can be viewed as an efficient low-complexity block entropy coding scheme having the much desirable properties of embeddedness, progressive transmission and fast encoding/decoding.

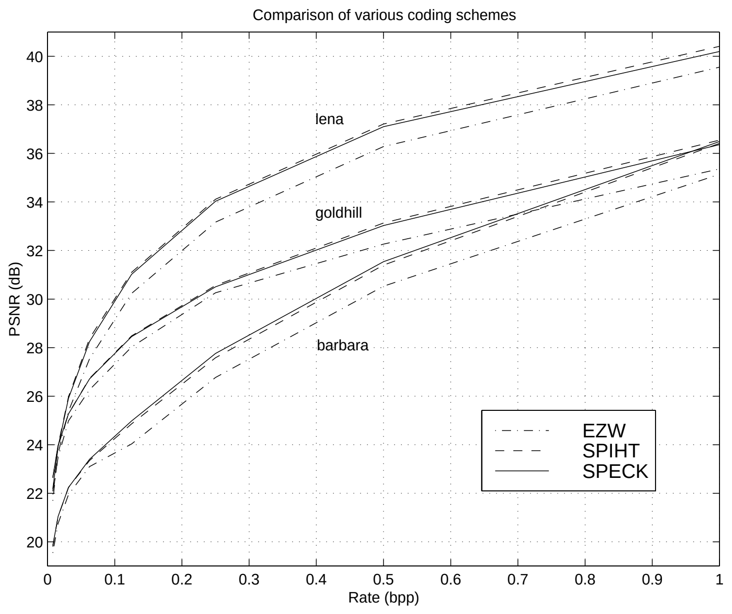

Figure 5. Comparative evaluation of the new coding method at low rates

Coding method	0.25 bpp	0.5 bpp	1.0 bpp
EZW	33.17 dB	36.28 dB	39.55 dB
AGP	34.10 dB	37.21 dB	40.38 dB
SPIHT	34.11 dB	37.21 dB	40.44 dB
SPECK	34.03 dB	37.10 dB	40.25 dB

Table 1. PSNR at various rates for Lena (512 x 512)

Coding method	0.25 bpp	0.5 bpp	1.0 bpp
EZW	26.77 dB	30.53 dB	35.14 dB
AGP	27.81 dB	31.61 dB	36.55 dB
SPIHT	27.58 dB	31.40 dB	36.41 dB
SPECK	27.76 dB	31.54 dB	36.49 dB

Table 2. PSNR at various rates for Barbara (512 x 512)

Coding method	0.25 bpp	0.5 bpp	1.0 bpp
EZW	30.31 dB	32.87 dB	36.20 dB
AGP	30.53 dB	33.13 dB	36.53 dB
SPIHT	30.56 dB	33.13 dB	36.55 dB
SPECK	30.50 dB	33.03 dB	36.36 dB

Table 3. PSNR at various rates for Goldhill (512 x 512)

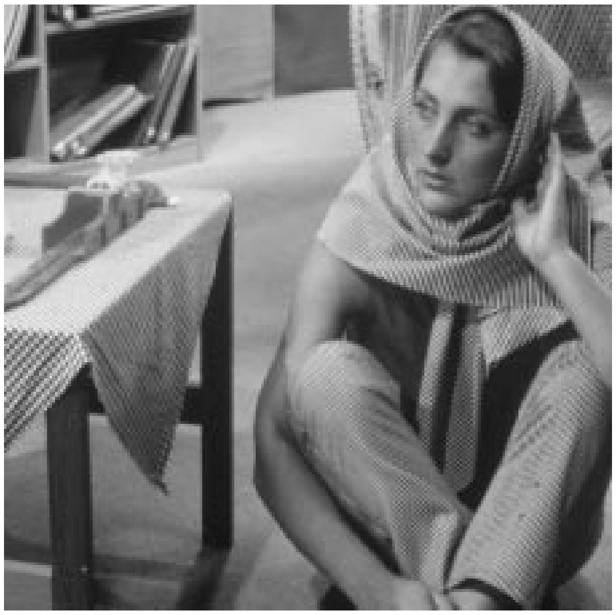
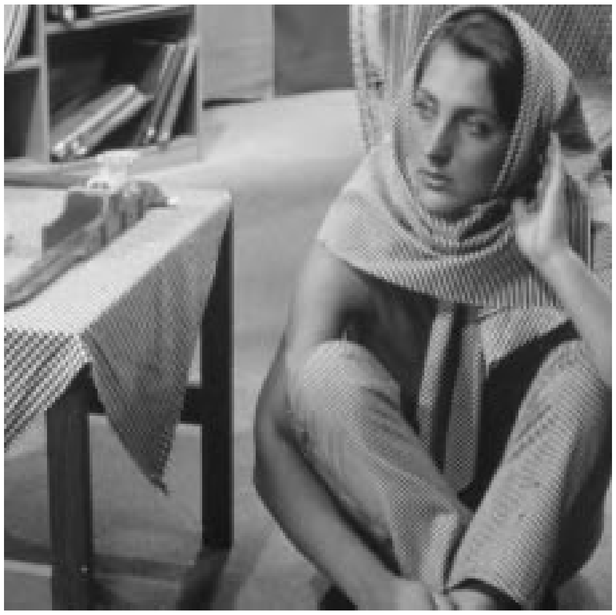

Figure 6. Coding results of SPECK on Barbara: Original (left), 1 bpp (right)

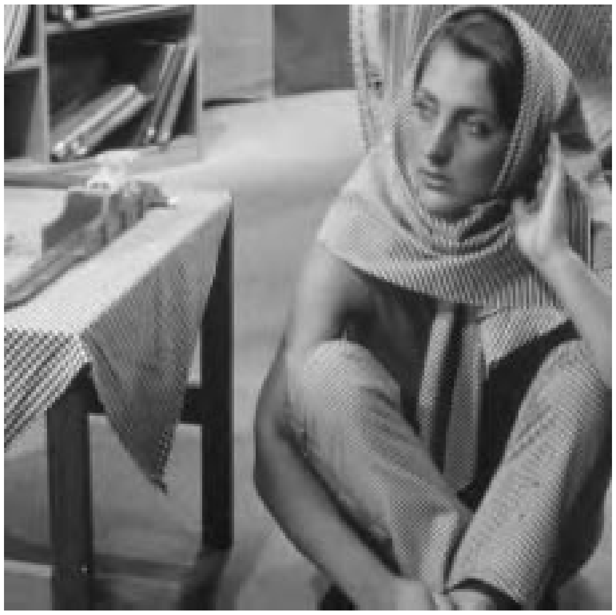
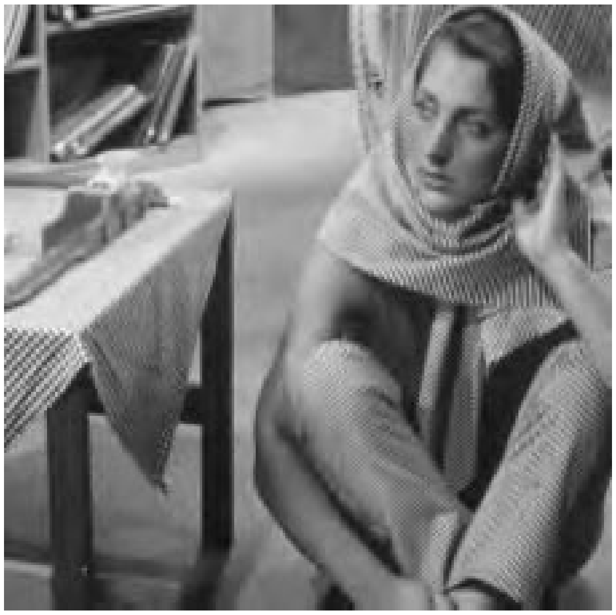

Figure 7. Coding results of SPECK on Barbara: 0.5 bpp (left), 0.25 bpp (right)

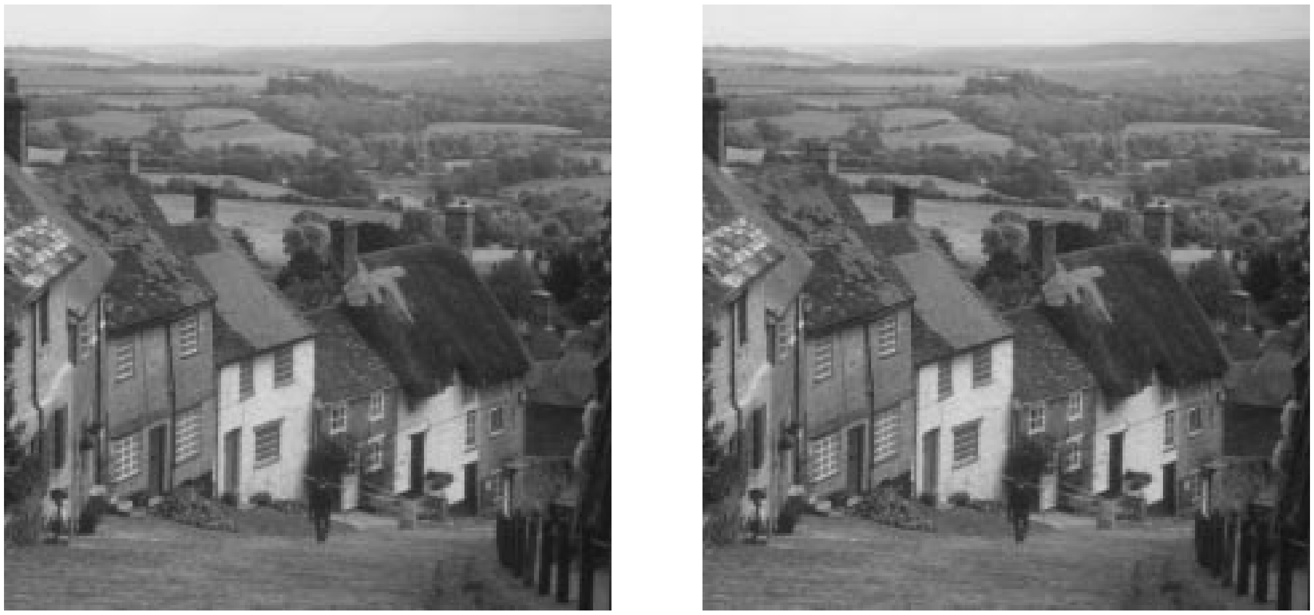

Figure 8. Coding results of SPECK on Goldhill: Original (left), 1 bpp (right)

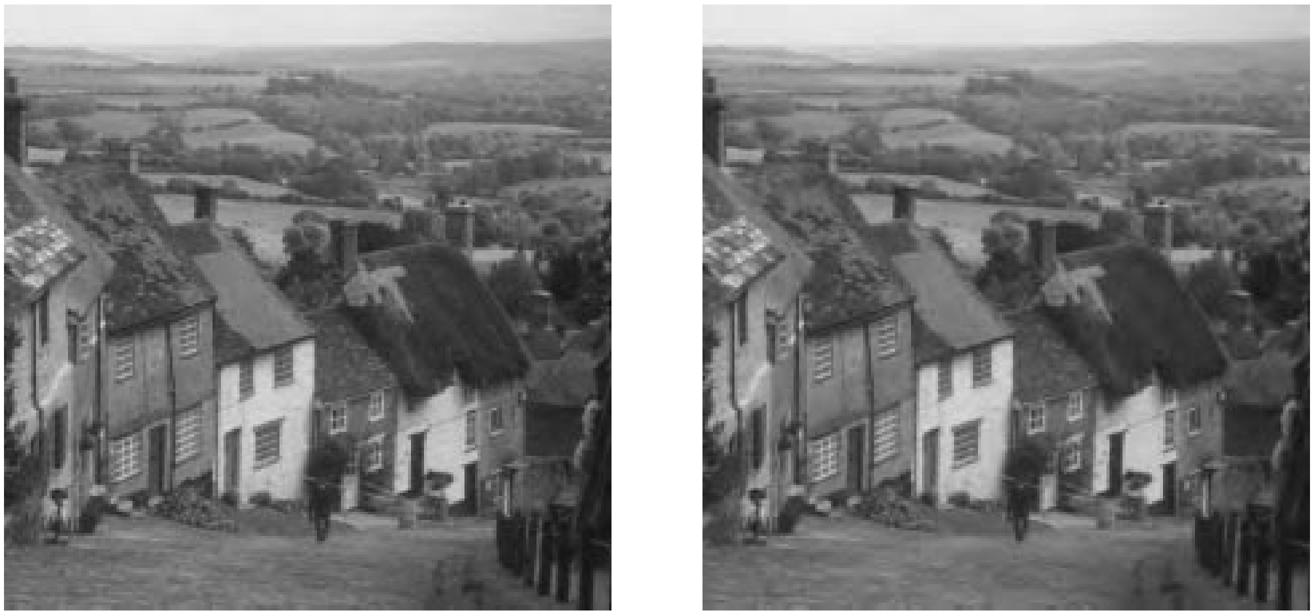

Figure 9. Coding results of SPECK on Goldhill: 0.5 bpp (left), 0.25 bpp (right)

ACKNOWLEDGMENTS

We would like to thank Charles Creusere of the Naval Air Warfare Center, Weapons Division, China Lake, CA, for supplying us with numerical results of the EZW algorithm on the Goldhill image.

REFERENCES

1. J. Shapiro, "Embedded image coding using zerotress of wavelet coefficients," *IEEE Trans. Signal Processing* **41**, pp. 3445–3462, Dec. 1993.
2. A. Said, "An improved zero-tree method for image compression," in *IPL TR-122, ECSE Dept., Rensselaer Poly. Inst., Troy, NY* , Nov. 1992.
3. A. Said and W. Pearlman, "Image compression using the spatial-orientation tree," in *IEEE Int. Symposium on Circuits and Systems, Chicago, IL*, pp. 279–282, May 1993.
4. A. Said and W. Pearlman, "A new, fast and efficient image codec based on set partitioning in hierarchical trees," in *IEEE Trans. Circuits and Systems for Video Technology*, vol. 6(3), pp. 243–250, June 1996.
5. T. Cover and J. Thomas, *Elements of Information Theory*, John Wiley & Sons, Inc., New York, 1991.
6. A. Said and W. Pearlman, "Low-complexity waveform coding via alphabet and sample-set partitioning," in *Visual Communications and Image Processing '97, Proc. SPIE 3024* , pp. 25–37, Feb. 1997.
7. J. Andrew, "A simple and efficient hierarchical image coder," *ICIP 97, IEEE Int'l Conf. on Image Proc.* **3**, pp. 658, paper no. 573, 1997.
8. M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing* **1**, pp. 205–220, April 1992.