IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 9, Issue 1, Ver. III (Jan. 2014), PP 36-47
www.iosrjournals.org

Highly Scalable Single List Set Partitioning in Hierarchical Trees Image Compression

Ali Kadhim Al-Janabi
Faculty of Engineering, University of Kufa-Iraq

Abstract: *A highly scalable image compression system produces a compressed image that can be easily decoded at several bit-rates (qualities) and resolutions (sizes). This is a very interesting feature in the heterogeneous environments of the Internet to satisfy the different user's requests. This paper proposes a new algorithm termed Highly Scalable-Single List-SPIHT (HS-SLS). The proposed HS-SLS algorithm produces a highly scalable bit-stream that is both rate and resolution scalable. The flexible bit-stream of the HS-SLS encoder can easily be adapted to various resolution requirements at any bit-rate using a very simple scaling process that is performed on-the-fly and without the need to decode the bit-stream. A unique feature of the new algorithm is that it has low memory requirement and low complexity due to its simple memory management. In addition, the size of the usable memory is fixed and thus can be predefined which avoids the dynamic memory allocation problem. These features make the HS-SLS algorithm very suitable for hardware implementation. The price paid for these valuable features is very slight decrement in the algorithm's performance as compared to the original SPIHT algorithm.*

Index Terms: *Highly Scalable Image Compression, Low Memory Image Compression, Rate scalability, Resolution scalability, Set Partitioning image Compression, SPIHT, Wavelet image compression, Zero-tree coding.*

I. Introduction

Conventional image compression schemes require that compression parameters, such as the requested image quality and display resolution (dimensions of the reconstructed image), be known at the time of compression. Due to the explosive growth of the Internet and networking technology, a huge number of users with different requirements and network access speeds (bandwidths) can easily communicate. In these diverse situations, it is difficult to predict the traffic on the network and the user request at the time of encoding. For example, mobile phones, handheld computers and desktop computers all have different display resolutions, bandwidth constraints, and processing capabilities. Therefore, there is an increasing demand for scalability to optimally service each user according to its available resources and request. Broadly speaking, scalability essentially means that the image is encoded in such a way that the compressed bit-stream can be manipulated or scaled in a simple manner in order to satisfy some constraint on bit-rate (rate scalability), display resolution (resolution scalability), or both (full scalability) without the need to transcode (decode and re-encode) the compressed bit-stream. A scalable image coder operates without prior knowledge of the bit-rate and/or the resolution at which the image will be decoded in the future. Therefore, the encoder must compress the image at the highest bit-rate and resolution. The compressed bit-stream is then stored on an image parser –which can be part of an image server. Different users with different requirements in terms of image qualities and display resolutions send their request to the parser, and the parser provides them with a properly scaled bit-stream. Scalability is an important feature to the extent that all emerging image and video compression standards such as the JPEG2000 image compression and MPEG-4 video compression are highly scalable [1, 2].

The Discrete Wavelet Transform (DWT) has gained a wide popularity in signal processing in general and in image compression in particular due to its interesting features such as good energy compaction capability, localization across time (or space) and frequency, and the ability to represent the image at several resolutions. Briefly speaking, the1Dimentional-Forward DWT (1D-FDWT) makes use of frequency selective two channel filter bank (low-pass and high-pass) to decompose the input signal into low and high frequency bands referred to as subbands. On the other hand, the 1D-Inverse DWT (1D-IDWT) makes use of another filter bank to reconstruct the input signal from the low and high frequency subbands. For image applications, the separable (2D) DWT is usually performed. The 2D-FDWT decomposes each row of the image using the filter-bank. The same filter-bank is then applied vertically to each column of the transformed image. The result is four subbands, termed LL_1 (horizontally and vertically low-pass), HL_1 (horizontally high-pass and vertically low-pass), LH_1 (horizontally low-pass and vertically high-pass), and HH_1 (horizontally and vertically high-pass). To increase the efficiency of the 2D-FDWT, multiple decomposition stages is recursively performed on the LL_1 subband because it is smoothed version of the original image and thus still highly correlated (i.e., LL_1 is decomposed to LL_2, HL_2, LH_2, and HH_2, and so on). *K*-level decomposition results in 3K+1 subbands that are organized into

K+1 resolution levels, R_0, $R_1 \ldots R_K$. The lowest resolution level, R_0, consists of the single LL_K subband. The next resolution level contains the three horizontal, vertical, and diagonal subbands required to reconstruct LL_{K-1} subband with twice the horizontal and vertical resolution during the 2D-IDWT. In general, level r ($0 < r \leq K$) contains the three subbands required to reconstruct LL_{K-r} subband. Here the original image is interpreted as an LL_0 of highest resolution. For example, for K = 3, $R_0 = \{LL_3\}$, $R_1 = \{LH_3, HL_3, HH_3\}$, $R_2 = \{LH_2, HL_2, HH_2\}$, and $R_3 = \{LH_1, HL_1, HH_1\}$, as shown in Figure (1) where every resolution level is represented by a different color. This wavelet decomposition is referred to as dyadic, octave or Mallat decomposition [3, 4].

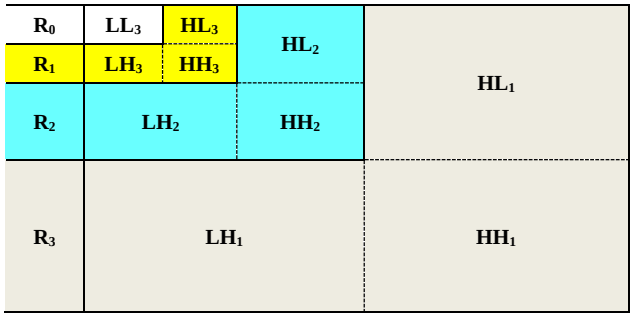


Figure (1): an image with 3 (2-D) DWT decomposition levels

A resolution scalable image compression system produces a bit-stream that contains distinct subsets representing each resolution level. The (2-D) dyadic DWT provides resolution scalability in natural way. The DWT's multi-resolution properties arise from the fact that the LL_k subband, $1 \leq k \leq K$, is a reasonable low resolution rendition of LL_{k-1} with half the width and height. To reconstruct the image at resolution r, $0 \leq r \leq K$, the subbands at R_0 through R_r are combined and then performing r stages of 2D-IDWTonly. For example, referring to Figure (1), the LL_3 subband would correspond to the lowest resolution for 3-level decomposition. The image can be reconstructed at resolution 1 by combining the LL_3 subband with the three subbands of R_1 (LH_3, HL_3 and HH_3), and performing single stage of 2D-IDWT to obtain LL_2. Continuing in this way, the image can be reconstructed at resolution 2 and 3 (the full size image). Therefore if the resolution levels are identifiable within the compressed bit-stream, the image can be reconstructed at any resolution level by simply decoding those portions of the bit-stream that contain the subbands that belong to that resolution and all the previous resolutions levels [4, 5].

A rate scalable image coder generates a bit-stream that consists of a set of embedded parts that offer increasingly better Signal-to-Noise Ratio (SNR). The rate scalable bit-stream (also known as an embedded bit-stream) can achieve the best possible image quality for the number of bits received at any truncation point. In other words, a rate scalable bit-stream can be truncated at any point while maintaining the best possible quality for the selected bit rate. To this end, the data with the highest distortion reduction per average bit should be coded as early as possible. It can be shown that a coefficient that has high magnitude has greater distortion reduction than a coefficient that has low magnitude, and for a given coefficient, its most significant bit (MSB) has greater distortion reduction than its least significant bit (LSB). Thus to obtain an optimal rate scalable bit-stream, the DWT image must be coded bit-plane by bit-plane rather than coefficient by coefficient and within each bit-plane, the bits that have the highest distortion reduction from all subbands should be coded first [4, 6].

A highly scalable image compression system produces a bit-stream that is both resolution and rate scalable, i.e., it retains the rate scalability feature for the requested image resolution only. As the target bit-rate and requested resolution are not known during compression, a highly scalable bit-stream requires to be structured in such a way that elements pertaining to each type of scalability are identifiable in the bit-stream, allowing easy extraction during the scaling process. Unfortunately, it is not easy to generate a highly scalable bit-stream. The difficulty arises because rate scalability interferes with resolution scalability. Notice that the main aim of rate scalability is to preserves information ordering by coding the bits of the coefficients that cause the highest distortion reduction first, regardless of their positions. This may lead to the fact that coefficients in a higher resolution level may be coded before coefficients in a lower resolution level, destroying resolution scalability. In resolution scalability, on the other hand, the coefficients are encoded in increasing order of resolution levels, regardless of their contribution to the total distortion, destroying rate scalability [2, 5].

II. Related Work

It is well known that the Embedded Block Coding with Optimized Truncation (EBCOT) algorithm, which is the core coding of the JPEG2000 standard, is the most efficient among all wavelet-based scalable image compression coders [4, 5]. Unfortunately the algorithm has high computational complexity due to using arithmetic coding, each coefficient is scanned three times in each bit-plane coding pass, and the compressed bit-

stream requires a time consuming rate distortion optimization process in order to obtain the intended highly scalable bit-stream. These factors make EBCOT very slow and difficult to attain in General Purpose Processor (GPP) or Digital Signal Processor (DSP) chips [2, 7].On the other hand, set portioning image compression algorithms such as the Set Partitioning in Hierarchical Trees (SPIHT)[8] and Set Partitioning Embedded bloCK (SPECK) [9] coders run by about (5-7) times faster and possess all the desirable features and sacrifice very little, if at all, in performance compared to JPEG2000 [10].These schemes employ some kind of significance testing for sets of pixels, in which the set is tested to determine whether the maximum magnitude in it is above a certain threshold. If the set is significant (SIG), it is partitioned; and if the set is insignificant (ISIG), it is represented by one symbol. The same process is recursively repeated for the SIG sets until all pixels are encoded. The main advantages of these schemes are they have relatively low computational complexity, good performance, and they generate a rate scalable compressed bit-stream. The price to be paid for these advantages is the high memory requirements as these algorithms use linked lists to keep track of which sets/pixels need to be tested. At high rates, the lists sizes may be bigger than the image size. In addition, the use of lists causes a variable data dependent memory requirement and the need for complex memory management due to the continual process of adding and removing nodes to and from these lists [11]. Finally, the generated bit-stream does not explicitly support resolution scalability and do not provide a bit-stream that can be easily reordered according to the resolution and rate desired by a decoder [5, 12].

Several works that added resolution scalability to the set partitioning schemes to obtain highly scalable bit-stream are presented in the literature. H. Danyali and A. Mertins [12] presented the Highly Scalable-SPIHT (HS-SPIHT) algorithm. It upgrades the original rate scalable SPIHT to produce highly scalable bit-stream. The HS-SPIHT adds the resolution scalability through the introduction of multiple resolution-dependent lists and resolution-dependent coding passes. The main advantage of the HS-SPIHT coder is the simplicity of bit-stream scaling process as no explicit rate allocation mechanism is needed. However, it has the same drawbacks of the original SPIHT in terms of high memory requirements. In addition, using the resolution dependent linked lists requires extra processing for memory managements. Moreover**,** HS-SPIHT needs a mechanism to distinguish the resolution level of each set in order to store it in the corresponding resolution dependent list. Clearly these requirements increase the algorithm complexity compared to the original SPIHT.

A. monauwer and K. Khan presented the Listless HS-SPIHT (LHS-SPIHT) [13]. It replaces the resolution dependent lists that are used with HS-SPIHT by state markers with an average of 4 bits/coefficient to keep track of pixels and set significance information. LHS-SPIHT uses the set partitioning rules similar to that of HS-SPIHT and produces almost identical bit-stream. The coder is very efficient in memory usage but unfortunately it has high computational complexity because it must process all the image pixels two times in each coding pass. The first time is in the Insignificant Pixel Pass to code the newly significant pixels and the second time is in the Refinement Pass to refine the pixels that are found significant in the previous coding passes. It worth to noting that SPIHT uses the lists for two main purposes: the first is for complexity reduction as only the elements stored in the lists are processed in each coding pass. The second purpose is to get better performance by preserving information ordering as the lists are processed in a specific order [5, 6].Therefore removing the lists will evidently increase the complexity of the algorithm and may decrease its performance. This means that LHS-SPIHT trades memory for complexity and efficiency i.e., it reduces memory usage at the cost of increasing complexity and decreasing performance.

In [14], we have proposed a novel coder, termed Single List SPIHT (SLS). The coder has low memory requirements as it needs about six times less memory than the original SPIHT. Like SPIHT, the SLS coder produces a rate scalable bit-stream. The theoretical analysis and experimental results showed that the proposed SLS algorithm has better performance and has nearly the same complexity as the original SPIHT. This means that we reduced the memory requirements by about six times and enhanced the performance of SPIHT without affecting its simplicity and without paying any additional overhead cost. In this paper we upgrade the SLS algorithm to produce a highly scalable bit-stream. The proposed algorithm is termed Highly Scalable-Single List SPIHT (HL-SLS). The new HL-SLS coder solves the scalability problem through the introduction of the resolution dependent parts list and the resolution dependent bit-plane coding passes. A unique feature of the HL-SLS is that while the other solutions have either higher memory requirements and/or higher complexity with respect to their rate scalable counterparts, the proposed HL-SLS has nearly the same complexity and the same memory requirements and management as the original SLS encoder. The remainder of the paper is organized as follows: section III gives brief review for the SPIHT and the SLS algorithms. Section IV introduces the new HS-SLS algorithm. Section V covers the simulation results of the algorithm. Finally, Section VI gives the concluding remarks of the paper and presents new directions for future work.

III. The SPIHT and the Single List SPIHT (SLS) algorithms

a) SPIHT

The SPIHT algorithm [8] exploits the self-similarity that exists between the wavelet coefficients that

belong to the subbands across the different resolution levels to form trees called Spatial Orientation Trees (SOTs). In the following descriptions the terms coefficient, node, and pixel will be used interchangeably. A SOT is defined in such a way that each node has either no children (the leaves at the highest level) or four children at the next level of similar orientation which always form a group of (2 × 2) adjacent pixels. The pixel at the lower level is called the parent, and the four children at the next level are called the offspring. The pixels in the lowest resolution level (the LL_K subband) of the DWT image are the tree roots and are also grouped into blocks of (2×2) adjacent pixels. However, their offspring branching rule is different; in each block, the top-left coefficient has no descendants, i.e. it is not part of any tree. Figure (2) depicts the SOTs for the first group of (2×2) pixels in subband LL_2 for a DWT image with 2 decomposition levels where each numbered color corresponds to a complete SOT that spans across the different resolution levels. The idea of the SPIHT algorithm is based on the following hypothesis: for any SOT, if its root is ISIG, then the probability of finding SIG descendants is low, i.e., the whole SOT is ISIG and thus can be coded by one symbol.

The SPIHT algorithm makes use of three linked lists, called List of Insignificant Sets (LIS), List of Insignificant Pixels (LIP), and List of Significant Pixels (LSP). As mentioned previously the lists are mainly used to reduce the algorithm's complexity and to preserve information ordering. In all lists each entry is identified by a coordinate (i, j), which in the LIP and LSP represents individual pixels, and in the LIS represents the roots of the SOTs. The roots may be either of type A or B. A root of type A represents the root of all descendants while a root of type B represents the root of all descendants excluding its offspring (the four direct children).

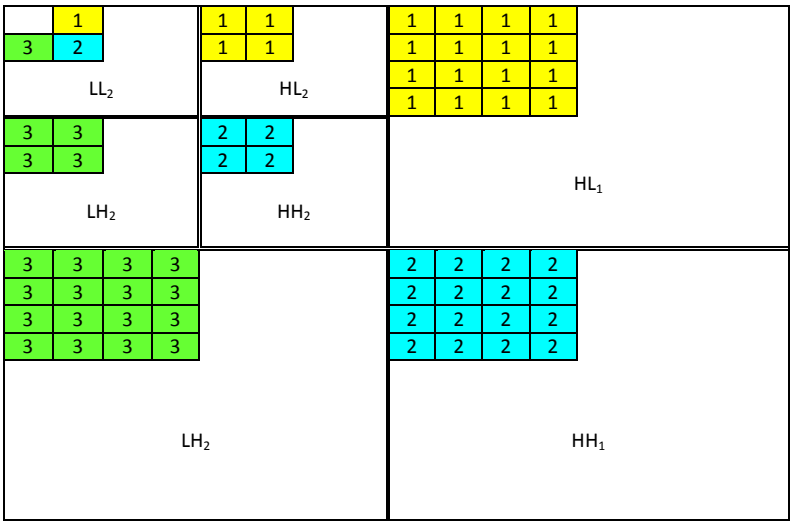


Figure (2): The SOTs for a DWT image with 2 decomposition levels

The SPIHT algorithm consists of initialization and several bit-plane coding passes; one for each bit-plane level. At initialization, it first computes the maximum bit-plane (n_{max}) based on the maximum value of the DWT image. n_{max} is given by:

$$\boldsymbol{n_{max} = \lfloor log_2 \left(max_{\forall (i,j) \in I} |c_{ij}| \right) \rfloor} \qquad \textbf{(1)}$$

where I is the DWT Image, c_{ij} is a wavelet coefficient at location (i, j), and $\lfloor x \rfloor$ is the nearest integer $\leq$ x. n_{max} is sent to the decoder within the bit-stream as a side information in order to start decoding at n_{max}. Then, it initializes the LIP with the coordinates of all the pixels in the LL_K subband, the LIS with all the coordinate of pixels in LL_K subband that have offspring as type A sets, and the LSP as an empty list.

Each bit-plane (except the first) is encoded by two passes: the sorting and the refinement. Only the sorting pass is made for the first bit-plane. The sorting pass identifies the new SIG pixels and the refinement pass codes the pixels that are found SIG at previous passes. For a given bit-plane n, a pixels c_{ij} is considered SIG with respect to n if

$$\boldsymbol{2^n \leq |c_{ij}| < 2^{n+1}} \qquad \textbf{(2)}$$

In the same way, a set of pixels is considered SIG with respect to n if the set contains one or more SIG pixels; otherwise it is ISIG.

During the sorting pass, each entry in LIP is tested for significance with respect to n. If the pixel is

ISIG, a (0) is transmitted to the compressed bit-stream. On the other hand, if the pixel is SIG, a (1) and a sign bit representing the sign of that pixel (e.g., 0 for positive and 1for negative pixel) are transmitted to the bit-stream, and the entry is *removed* from LIP and *added* to *end* of LSP to be refined at the next bit-plane passes. Next, each set in LIS (which represent the root of an SOT) is tested as follows: If the set is of *type* **A**, its SOT is tested for significance. If the SOT is ISIG, a (0) is transmitted (i.e., the whole SOT is coded by a single bit). Otherwise (the SOT is SIG), a (1) is transmitted, and each one of the set's four children is tested. If the child is SIG, a (1), and its sign bit are transmitted to the bit-stream, and its coordinates are *added* to the end of LSP. Otherwise (the child is ISIG), a (0) is transmitted and its coordinates are *added* to LIP to be tested in the next bit-plane passes. Finally, if the set has grandchildren, it is *moved* to the end of LIS as a type B set. On the other hand if the set is of *type* **B** and has at least one SIG child, a (1) is transmitted, the set is *removed* from LIS, and its four children are *added* to the end of LIS as type A sets to be tested in the current pass. Otherwise (all the set's children are ISIG), a (0) is transmitted and the set remains in LIS to be tested in next bit-plane passes.

In the refinement pass, all pixels in the LSP, except those which have been added to it during the current pass, are refined by outputting its n_{th} MSB to the bit-stream. After finishing the sorting and refinement passes for bit-plane *n*, *n* is decremented by 1 to begin coding the next bit-plane.

The SPIHT algorithm has the following drawbacks [11, 12]:

1. It needs a huge amount of memory due to using the lists. More precisely the LIP and LSP lists which store the (i, j) coordinates of individual pixels dominate the total storage. The maximum number of entries in each list is equal to the number of image pixels. On the other hand, the LIS has less memory requirement because it stores the (i, j) coordinates of the roots of the SOTs. In addition, the roots in the highest level (LH_1, HL_1, and HH_1subbands) are not stored because they don't have descendants. The size of these subbands is 3/4 the image size. Thus the maximum number of entries of LIS is one-fourth of image pixels. Therefore, the working memory of SPIHT is at least 2.25 times the image size.
2. It has complex memory management due to the removing and adding nodes to the LIP and LIS lists.
3. The lists sizes cannot be pre-allocated because the number of list entries can't be determined in advance as it depends on the compression bit-rate and on the amount of image details. This problem can solved by using either dynamic memory allocation or pre-allocating the lists to the maximum size. Evidently, the former solution slows the algorithm, and the latter one increases its memory requirements.
4. It produces a rate scalable bit-stream that does not explicitly support resolution scalability and thus cannot be easily reordered according to the rate and resolution desired by the decoder.

b) SLS

The SLS algorithm [14] overcomes the huge memory requirements and the complex memory management drawbacks of the original SPIHT algorithm (the first three drawbacks). In addition, the SLS coder has slightly better performance than SPIHT. The basic idea of the SLS is based on the fact that the LIP and LSP are constructed from the offspring (the four children) of a root that belongs to a SIG SOT. So, instead of storing the offspring in these lists, they can be recomputed in each pass. Evidently, this will increase the complexity of the algorithm. However, this complexity increment is compensated by the reduced memory management overhead (as shown shortly). The SLS algorithm replaces the LIP and LSP lists (which dominate the total memory usage) by two bits state marker. In other words, each pixel is provided by two status bits referred to as σ to determine the type of the pixel as follows:

- $\sigma = 0$: New ISIG Pixel (NIP) is an ISIG pixel that is not yet tested.
- $\sigma = 1$: Visited ISIG Pixel (VIP) is a pixel that is tested ISIG in the previous passes.
- $\sigma = 2$: New SIG Pixel (NSP) is a pixel that just becomes SIG in the current bit-plane pass.
- $\sigma = 3$: Visited SIG Pixel (VSP) is a pixel that is tested SIG in the previous passes.

In addition, the SLS encoder makes use of a single list termed the List of Root Sets (LRS). Like the LIS, the LRS also stores the (i, j) coordinates of the roots of the SOTs. So the LRS has exactly the same memory size as the LIS (one-fourth the image size). However, the adopted coding method makes it possible to implement the LRS as a simple 1-D array that is accessed sequentially in First In First Out (FIFO) manner because once a set is added to the LRS, it will be never removed. In contrast, the LIS used with the SPIHT coder must be implemented as a linked list that is accessed randomly due to the continual process of adding and removing nodes to and from it. Evidently, implementing the LRS as a 1-D array together with removing the LIP and LSP will greatly simplify the memory management problem. Moreover, the type field used with the LRS is used differently. In the LRS A set of type A is untested set or a set that has no SIG SOT and a set of type B has one or more SIG SOT whereas in the LIS a set of type A represents the root of all descendants and set of type B represents the root of all descendants excluding the four direct offspring.

The SLS algorithm consists of an initialization stage and several bit-planes coding passes. At initialization, it first computes and outputs the maximum bit-plane n_{max} to the bit-stream, and set n to n_{max}. Then it stores the (i, j) coordinates of the pixels in LL_K subband that have offspring in the LRS as type A sets. The first bit-plane pass starts by coding LL_K subband which consists of New ISIG pixels (NIPs) only. Each NIP is tested for significance with respect to n. If it is still ISIG, a (0) is outputted to the bit-stream and it is updated to VIP. If it becomes SIG, a (1) and its sign bit are outputted to the bit-stream and it is updated to NSP. Next, the sets of the LRS (which are sets of type A) are sequentially processed. For each set, its SOT is computed and then tested for significance with respect to n. If the SOT is still ISIG, a (0) is outputted (i.e., the whole SOT is represented by a single bit). On the other hand, if the SOT is SIG, a (1) is outputted, the set is updated to type B set, and each one of the set's offspring (which is an NIP) is coded and updated as given above. Then, if the set has grandchildren (the set's offspring aren't in the highest resolution level), the offspring are added to the end of LRS as type A sets to be coded in the current bit-plane coding pass. Finally n is decremented by 1 to start a new bit-plane coding pass.

Each one of the next bit-plane coding passes also starts by coding the pixels in LL_K. At these passes LL_K may contain VIPs and/or NSPs only. A VIP is coded exactly in the same way of coding the NIP except that there is no need to update it if it is still ISIG because it is already a VIP. On the other hand, a NSP is updated to VSP and then is refined by outputting its n_{th} MSB to the bit-stream. Next, the LRS list is scanned two times. In the first scan, only the sets of type B are inspected. For each set, its four offspring is computed which may be of type VIPs and/or NSPs. A VIP is coded as given above whereas a NSP is updated to VSP to be refined later in the second scan pass. In the second scan, all the sets in LRS are inspected. If the set is of type A, it is processed exactly in the same way as was done in the first coding pass. On the other hand, if the set is of type B, only the VSPs of the set's four offspring are refined by outputting its n_{th} MSB bit to the bit-stream. Finally n is decremented to start a new bit-plane coding pass.

IV. The Proposed Highly Scalable-SLS (HS-SLS) Algorithm

a) HS-SLS Description

Broadly speaking, a rate scalable bit-stream can be upgraded to a highly scalable one, i.e., a bit-stream that is both rate and resolution scalable if the coding information in each bit-plane coding pass are arranged in increasing order of resolution levels. In addition, each level must be distinguishable in the output bit-stream, allowing easy access to the data needed to reconstruct the image at the desired spatial resolution during the scaling process. In this section, we present the Highly Scalable-SLS (HS-SLS) algorithm. The proposed algorithm is transparent in the sense that the transition to full scalability is done by only separating and identifying the resolution levels within each bit-plane coding pass. A unique feature of HS-SLS is that it adds the resolution scalability without the need of using the multiple resolution dependents lists as was done in HS-SPIHT [12]. Moreover, it doesn't need to process all the image pixels twice in each bit-plane pass as was done in [13]. This means that HS-SLS doesn't involve any increase in complexity and memory requirements and management as compared to the original SLS.

The HS-SLS coder is based on three facts. The first fact is that the once a set is added to LRS, it will be never removed. Thus if the sets are added to LRS according to a specific order, this order will be preserved during the entire coding process. This means that if the sets are stored in increasing order of resolution levels, this order will be always preserved. The second fact is that the number of pixels in each resolution level is fixed. Let N_r denote the number of pixels in resolution level R_r. since R_0 consists of LL_K subband only, then $N_0 = (LL_K \times LL_K)$. For $r = 1, 2 \ldots K$, N_r is given by:

$$\boldsymbol{N_r = 3(LL_{k+1-r} \times LL_{k+1-r})} \qquad \textbf{(3)}$$

For example, for K = 3, the number of pixels in R_0 is $(LL_3 \times LL_3)$, and the number of pixels in R_1 is $3(LL_3 \times LL_3)$ and so on. Therefore the sets of each resolution level (except the highest since it has no descendants) can be stored in a specific fixed size part in the LRS. Thus, the LRS can be thought to consist of K parts which are referred to as Resolution Part (RP). RP_r has size N_r and stores the sets that belong to level r. Figure (3) depicts the adopted Resolution Parts within the LRS for K = 3.

RP_0	RP_1	RP_2
$(LL_3 \times LL_3)$	$3(LL_3 \times LL_3)$	$3(LL_2 \times LL_2)$

Figure (3): The adopted resolution parts of the LRS for K = 3

The third fact is that for a set at any resolution level (except the highest), its offspring are located at the next level. Therefore, these offspring can be easily stored in straight forward manner in its assigned resolution part

without the need of extra processing and without the need to use any additional markers to distinguish the level to which the set is belongs as is done in HS-SPIHT and LHS-SPIHT algorithms. For instance, a set at level R_0, its offspring are located at R_1 and are stored in RP_1 and so on. In order to keep track of the sets that are added to the different resolution parts, HS-SLS employs two small Resolution Index Tables termed the Start Index Table (SIT) and the End Index Table and (EIT) of size K each. SIT[0] and EIT[0] are both initialized to 0, and SIT[r] and EIT[r], $0 < r < K$, are initialized to N_{r-1} which is the maximum number of sets in level R_{r-1}. EIT[r] is then incremented each time a set is added to part RP_r. Remember that the LRS is initialized by the (i, j) coordinates of the pixels of R_0 (the LL_K subband) that have offspring as type A sets. Evidently these sets are stored in RP_0 and hence EIT[0] is updated to ¾ N_0 because the top-left pixel of every (2×2) pixels has no descendent.

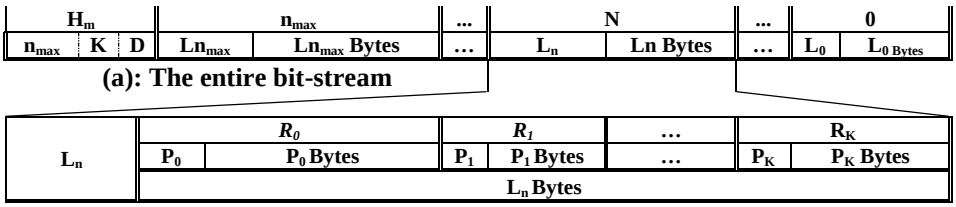
The HS-SLS works exactly as SLS except that it makes use of resolution dependent coding passes with the associated LRS resolution parts. That is for each bit-plane pass, the HS-SLS codes all the information that belongs to R_0 and proceeds to R_1, R_2 and so on. This can be very easily attained by dealing with the resolution parts of the LRS instead of the entire LRS and putting sufficient markers within the bit-stream to identify the information that belongs to the different resolution levels. In order to clarify the adopted coding method, remember that each bit-plane coding pass starts by coding the LL_K subband and then it processes the LRS. Since LL_K represents R_0, hence each pass starts coding the pixels in R_0. Next, the *resolution parts* of the LRS are processed *sequentially*. Every resolution part PR_r is scanned two times. In each scan, all the sets in PR_r that are *determined* by SIT[r] through EIT[r] are inspected. In the first scan, only the sets of type B are coded in the same way as was done with SLS. In the second scan, all the sets in PR_r are inspected. If the set is of type B, only the VSPs of the set's four offspring are refined as given before. On the other hand, if the set is of type A, it is tested for significance. If it is still ISIG, (0) is outputted and nothing is done. If the set is tested SIG, the set's type is *updated* to type B set, and each one of the set's four offspring is *coded* and its status is *updated* as given before. Then, if the set has *grandchildren* (its offspring isn't in the highest resolution level), each one of its four offspring is *added* to its *location* in RP_{r+1}which is *determined* by EIT[r+1] as type A sets and EIT[r+1] is *incremented* correspondingly. In this way, LRS will stores the sets in increasing order of resolution levels and consequently every bit-plane coding pass will also be coded in increasing order of resolution levels.

As shown, the proposed HS-SLS produced the highly scalable bit-stream simply by arranging the data in increasing order of resolution levels and without any need to additional memory management overhead. In contrast, the HS-SPIHT coder [12] has complex memory management due to dealing with multiple linked lists and each list is accessed randomly as the nodes are added and removed to and from it continuously. That is, HS-SPIHT deals with a set of LIP, LSP and LIS lists for each resolution level r (LIP_r, LSP_r, and LIS_r). Moreover, HS-SLS exhibits a slight complexity increment with respect to SLS because the two coding methods differ only by the way the sets of the LRS are ordered. Since the SLS coder has a very slight complexity increment with respect to SPIHT [14]. Therefore, the HS-SLS algorithm has also very slight complexity increment with respect to SPIHT. In contrast, the LHS-SPIHT algorithm [13] has much higher complexity than SPIHT because, as stated before, LHS-SPIHT must process all the image pixels two times in each coding pass. These features are unique to the proposed HS-SLS algorithm. Unfortunately, the HS-SLS coder exhibits some performance degradation with respect to SLS because the output bits within each bit-plane layer are ordered according to the resolution levels and not according to their Rate Distortion (R-D) properties. More specifically, it processes all the pixels in resolution part PR_r before proceeding to the next part PR_{r+1}. This may leads to code pixels that have lower R-D properties before pixels that have higher R-D properties leading performance deterioration. It should be noted that the HS-SPIHT and LHS-SPIHT also suffer from this limitation.

b) Bit-stream Formation and Scaling

The structure of the bit-stream generated by the HS-SLS encoder is shown in Figure (4). It consists of main header (H_m) and body of the bit-stream. The main header H_m contains the maximum bit-plane (n_{max}), the number of decomposition levels (K), and data (D) field such as the image name, image dimension, etc. The bit-stream consists of (n_{max}+ 1) layers. Each layer corresponds to bit-plane level. Each bit-plane layer consists of the layer length tag (L_n) followed by the layer body. L_n identifies the contribution in bytes of layer n to the total bit-stream. The body of the layer consists of (K+1) parts, one for each resolution level. Each resolution part consists of a length tag (P_r) that indicates the number of bytes that belong to resolution level r. It should be noted that all header information at the beginning of each bit-plane part is used solely by the image parser and does not need to be sent to the decoder. However, in order to improve the error resilience, these markers may be kept within the bit-stream.

The HS-SLS bit-stream can easily be reordered for multi-resolution decoding at any desirable bit rate. An image at resolution r and bit-rate B bit per pixel (bpp) can be reconstructed from the bit-stream by keeping the resolution levels 0, 1…r and discard all other levels, i.e. levels r+1, r+2…K, in each bit-plane layer. The procedure is continued for each layer until the bit-rate equals to the requested bit-rate, B.



(b): The bit-stream for bit-plane layer *n*

Figure (4): General structure of HS-SLS bit-stream

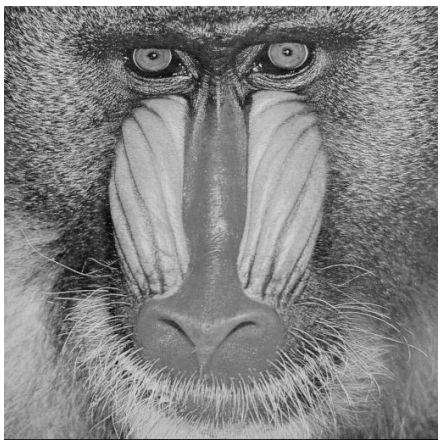
V. Experimental Results and Discussion

The proposed HS-SLS algorithm is evaluated by Borland C++ programming language version 5.02 using a PC equipped with Intel Core i3@1.8 GHz CPU and 2 GB RAM. The test is performed to the popular gray-scale (512×512) pixels 'Lena', 'Barbara', 'Baboon' (or Mandrill), and 'Goldhill' test images shown in Figure (5). Each image is transformed using the bi-orthogonal 9/7 Daubechies (2-D) DWT with 5 dyadic decomposition levels and the transform coefficients are rounded to the nearest integer prior to coding. The results of HS-SLS are compared with the SPIHT, SLS, HS-SPIHT and LHS-SPIHT algorithms. The results are represented by the algorithm's performance, its computational complexity, and its memory requirements.

(a) Lena

(b) Barbara

(c) Goldhill



(d) Baboon

Figure (5): Gray-scale 8 bpp (512×512) pixels test images

a) Performance

The performance is measured by the Peak Signal to Noise Ratio (PSNR) versus the compression bit-rate (the average number of bits per pixel (bpp) for the compressed image). PSNR is given by:

$$PSNR = 10\, log_{10} \frac{MAX^2}{MSE}\ dB \qquad (4)$$

Where *MAX* is the maximum value of the pixel and MSE is Mean-Squared Error between the original and the reconstructed images defined as:

$$MSE = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} [I_o(i,j) - I_r(i,j)]^2 \qquad (5)$$

where I_o is the original image, I_r is the reconstructed image, and M×N is the image size (number of pixels). Evidently, smaller MSE and larger PSNR values correspond to lower levels of distortion.

Tables (1) show the PSNR versus bit-rate for the SPIHT, SLS, LHS-SPIHT, and the proposed HS-SLS algorithms at full resolution (size) images. The results of the SPIHT are obtained by implementing the Public License MATLAB program presented by Dr. Mustafa Sakalli and Dr. William A. Pearlman [15] using the same test images at full scale. It is worth to noting that the SPIHT and SLS algorithms produce rate scalable bit-streams while the HS-SPIHT, LHS-SPIHT and HS-SLS algorithms produce highly scalable bit-streams. However, the results of SPIHT and SLS are presented because they represent the upper bound of HS-SPIHT, LHS-SPIHT, and HS-SLS respectively. In other words, at any bit-rate, the PSNR of HS-SPIHT and LHS-SPIHT is less than that of SPIHT, and the PSNR of HS-SLS is less than that of SLS. This is because, as mentioned before, the highly scalable algorithms sacrifice information ordering and because their bit-streams have auxiliary markers to identify the contribution of each resolution level in every bit-plane coding pass. Table (1) depicts the following:

- The SLS has slightly better PSNR than SPIHT for 'Lena' and 'Barbara' images and it has comparable PSNR for 'Baboon' and 'Goldhill' images. This indicates that SLS is competitive with the state-of-art rate scalable image compression algorithms.
- The HS-SLS has slightly lower PSNR than SLS. This is expected since HS-SLS doesn't preserve the information ordering as the pixels in each bit-plane coding pass are coded in increasing order of resolution levels and not according to their rate distortion (R-D) properties. Notice that this constraint is also found in the HS-SPIHT and LHS-SPIHT algorithms.
- The PSNR of HS-SPIHT and LHS-SPIHT are higher than that of SPIHT for 'Lena'. This is not possible due to the above reasons. However, this may be occurred due to using another version of the image.
- The PSNR of the proposed HS-SLS and the LHS-SPIHT are very close for 'Lena' and 'Baboon images' while HS-SLS gives better PSNR for 'Barbara' and 'Goldhill' images especially at low bit-rates. This indicates that the proposed HS-SLS algorithm has also comparable PSNR performance with the state-of-art highly scalable image compression algorithms.

Table (1): PSNR vs. Bit-rate at full Resolution (K = 5, and *r* = 5)

	PSNR (dB)									
	SPIHT	HS-SPIHT	LHS-SPIHT	SLS	HS-SLS	SPIHT	HS-SPIHT	LHS-SPIHT	SLS	HS-SLS
Bit Rate (bpp)	Lena					Barbara				
0.0625	27.25	27.52	26.85	27.30	27.17	23.31	22.97	22.59	23.33	23.29
0.125	29.39	30.31	29.93	30.01	29.82	24.03	24.12	23.65	24.61	24.24
0.25	32.71	33.33	33.19	32.98	32.86	26.92	26.68	26.75	27.28	26.86
0.5	36.13	36.57	36.49	36.26	36.07	30.88	30.53	30.48	31.10	30.81
1	39.50	39.93	39.58	39.61	39.42	36.06	35.28	35.19	36.20	35.72
Bit Rate (bpp)	Baboon					Goldhill				
0.0625	20.20	20.19	20.38	20.24	20.26	26.17	26.27	26.26	26.15	26.13
0.125	21.28	21.45	21.25	21.27	21.20	27.78	28.02	27.50	27.77	27.69
0.25	22.65	22.86	22.66	22.70	22.70	29.75	30.19	29.39	29.88	29.85
0.5	24.83	24.19	24.60	24.75	24.55	32.30	32.40	32.10	32.28	32.03
1	28.29	28.50	28.30	28.29	27.93	35.57	35.67	35.54	35.43	35.33

An image at resolution r, $0 \le r \le$ K, has $(1/2^{K-r}) \times (1/2^{K-r})$ the size of the original image. Therefore, a direct application of PSNR equation is not possible for $r <$ K because the original and the recovered images don't have the same size. However, to provide numerical results of the highly scalable algorithms when the recovered image has lower resolution than the original image, the same method adopted in [12] is used. The method is based on the fact that an image at resolution r corresponds to the LL_{K-r} subband. So, the original and reconstructed LL_{K-r} subbands can be compared instead of the original and reconstructed images. In this case, MAX and (M×N) represent the maximum pixel value and the size for the given LL_{K-r} subbands respectively. The MAX value for an 8 bpp gray-scale image is equal to $(255\times2^{K-r})$. This is due to the fact that resolution level r is obtained from the original image after applying $(K-r)$ levels of 2-D wavelet decomposition with filters having a DC amplification of $\sqrt{2}$ [4]. For example, when a 512×512 image is decomposed with K = 5, a reconstructed image at r = 5 has the full size and MAX = 255. On the other hand, at r = 4, the reconstructed image has 256×256 pixels and MAX = 510. The bit-rates for all levels are calculated according to the number of pixels in the original full size image. This enables not only to compare the results obtained for a given resolution at different bit rates but also to compare the results related to different spatial resolutions at a given coding budget. Tables (2-4) show the results of the LHS-SPIHT, HS-SPIHT, and the proposed HS-SLS algorithms at 1/2, 1/4, and 1/8 resolutions respectively. These tables demonstrate the PSNR superiority of the proposed HS-SLS algorithm over the LHS-SPIHT algorithm for all scales and bit-rates. The only exception is for the image 'Goldhill' at 1/8 resolution. Moreover, HS-SLS is better than HS-SPIHT for 'Barbara' and 'Babbon' images which are more complex than 'Lena' and 'Goldhill'' despite of the huge memory requirements and the complex memory management of the HS-SPIHT algorithm.

Table (2): PSNR vs. Bit-rate at 1 / 2 Resolution (K = 5, and *r* = 4)

	PSNR (dB)											
	HS-SPIHT	LHS-SPIHT	HS-SLS	HS-SPIHT	LHS-SPIHT	HS-SLS	HS-SPIHT	LHS-SPIHT	HS-SLS	HS-SPIHT	LHS-SPIHT	HS-SLS
Bit Rate	Lena			Barbara			Baboon			Goldhill		
0.0625	28.78	27.98	28.29	25.94	25.48	26.66	21.47	21.28	22.77	27.79	27.36	27.62
0.125	32.34	31.84	31.92	28.41	27.93	29.17	23.65	23.45	24.24	30.81	29.78	30.01
0.25	37.44	37.21	36.71	32.64	32.42	33.73	28.79	28.59	29.33	33.86	32.97	33.51
0.5	43.82	43.52	43.61	39.19	38.97	39.93	31.41	31.21	32.48	38.81	38.51	38.63
1	53.25	53.17	53.73	50.12	50.02	51.13	39.42	39.23	43.37	49.97	49.73	49.77

Table (3): PSNR vs. Bit-rate at 1 / 4 Resolution (K = 5, and *r* = 3)

	PSNR (dB)											
	HS-SPIHT	LHS - SPIHT	HS-SLS	HS-SPIHT	LHS - SPIHT	HS-SLS	HS-SPIHT	LHS - SPIHT	HS-SLS	HS-SPIHT	LHS - SPIHT	HS-SLS
Bit Rate	Lena			Barbara			Baboon			Goldhill		
0.0625	32.38	31.86	32	30.83	29.89	31.89	25.40	25.21	27.76	31.48	30.86	31.30
0.125	40.03	39.53	39.63	36.55	35.83	37.75	31.49	31.31	32.30	37.13	36.12	36.82
0.25	50.73	50.30	50.56	47.34	46.12	48.07	40.98	40.82	43.39	47.96	46.87	47.51
0.5	70.52	70.51	70.65	71.16	70.89	71.24	60.05	59.97	64.77	70.91	70.71	70.83
1	-	-	-	-	-	-	-	-	-	-	-	-

Table (4): PSNR vs. Bit-rate at 1 / 8 Resolution (K = 5, and r = 2)

	PSNR (dB)											
	HS-SPIHT	LHS-SPIHT	HS-SLS	HS-SPIHT	LHS-SPIHT	HS-SLS	HS-SPIHT	LHS-SPIHT	HS-SLS	HS-SPIHT	LHS-SPIHT	HS-SLS
Bit Rate	Lena			Barbara			Baboon			Goldhill		
0.0625	45.29	44.59	45.68	42.89	41.92	43.80	37.32	37.21	42.03	44.53	43.85	42.03
0.125	68.10	67.51	68.60	66.41	65.94	66.86	60.23	60.12	66.03	68.10	67.80	66.03
0.25	80.19	79.19	79.78	81.53	75.34	76.78	75.74	75.54	77.88	78.83	78.68	78.56
0.5	-	-	-	-	-	-	-	-	-	-	-	-

b) Complexity

As mentioned previously, the HS-SLS involves a slight complexity increment with respect to SLS because HS-SLS need very little extra processing for encoding the resolution levels in increasing order at the encoder and for the scaling process and for decoding the resolution level in increasing order at the decoder. On the other hand, the LHS-SPIHT has much higher complexity than SPIHT because the latter needs, in addition the above tasks, to process all image pixels two times per bit-plane coding pass. Table (5) shows the complexity of the algorithms represented by the CPU processing time (measured in milliseconds (msec)) required by the algorithm to encode and to decode the full scale "Lena" image at several bit-rates. As it can be clearly shown, the proposed HS-SLS runs faster than LHS-SPIHT.

Table (5): Processing time vs. Bit-rate for "Lena" image

	Coding time (msec)				Decoding time (msec)			
Bit-Rate (bpp)	**SPIHT**	**LHS-SPIHT**	**SLS**	**HS-SLS**	**SPIHT**	**LHS-SPIHT**	**SLS**	**HS-SLS**
0.125	**15**	**24**	**16**	**18**	**5**	**11**	**7**	**11**
0.25	**20**	**28**	**22**	**23**	**10**	**17**	**11**	**15**
0.5	**26**	**33**	**28**	**30**	**20**	**28**	**22**	**26**
1	**31**	**38**	**32**	**35**	**25**	**33**	**26**	**30**

c) Memory requirements

The memory requirement is measured by the amount of computer memory needed by the algorithm to compress the image at a given bit-rate. It is worth to noting that the HS-SPIHT can use variable size lists whose sizes depends on the image size and compression rate. However, this necessities the use of dynamic memory allocation which in turn slows the algorithm. In order to avoid this problem, the lists are assigned the maximum size. The maximum memory of HS-SPIHT is the same as SPIHT which is given by [11]:

$$MEM_{SPIHT}^{max} = b\left(\frac{9MN}{32}\right) \quad \textbf{Bytes} \qquad (6)$$

where b is number of bits needed to store the pixel (i, j) coordinates, $b = \lceil \log_2(M) \rceil + \lceil \log_2(N) \rceil$. The LHS-SPIHT uses a fixed memory with an average of 4 bits/pixel with total memory of (M × N)/2 Bytes. Finally, the HS-SLS also uses fixed memory which is equals to the maximum memory of SLS which is given by [14]:

$$MEM_{SLS} = b\left(\frac{MN}{32}\right) + \frac{MN}{4} \quad \textbf{Bytes} \qquad (7)$$

For instance, for gray-scale full resolution image of size (512×512), $b = 2 \times \log_2(512) = 18$ bits. Hence the total memory required by HS-SPIHT, LHS-SPIHT and HS-SLS are 1160 KB, 128 KB and 208 KB respectively. The HS-SLS has only 80 KB of memory increment with respect with LHS-SPIHT. However, HS-SLS still requires much less memory than HS-SPIHT.

VI. Conclusions

In this paper we presented the Highly Scalable Single List SPIHT (HS-SLS). The proposed HS-SLS algorithm extends the original rate scalable SLS coder successfully to a highly scalable scheme that supports combined resolution and rate scalability. This is achieved simply by arranging the data in increasing order of resolution levels using resolution-dependent coding passes and resolution-dependent parts list. The flexible, fully scalable bit-stream of the HS-SLS algorithm can easily be parsed to provide rate embedded bit-streams for all lower spatial resolution decoding using a very simple scaling process that is implemented on-the-fly and without the need to decode the main bit-stream. As shown from the experimental results, the HS-SLS encoder has better PSNR and lower complexity than LHS-SPIHT. The proposed scheme can be utilized in many applications such as image storage and retrieval systems, and multimedia information transmission, especially over heterogeneous networks where a wide variety of users need to be differently serviced according to their network access and data processing capabilities. In addition, the fixed size memory usage makes the HS-SLS algorithm very suitable for hardware implementation. Finally, The HS-SLS algorithm is very useful for volumetric and 3-D image compression systems benefiting from its reduced memory and simplicity.

References

[1] Salomon D., "**Data Compression: the Complete Reference**," third 3rd ed., Springer, 2004

[2] D. Taubman and M. Marcellin, "**JPEG2000: Image Compression Fundamentals, Standards, and Practice**", 2nd ed. Norwell, MA: Kluwer, 2002.

[3] Vetterli M. and Kovaccevic J., "**Wavelets and Subband Coding**," Prentice-Hall, New Jersey, 1st edition, 1995.

[4] Rabbani M. and Joshi R., "**An Overview of the JPEG 2000 Still Image Compression Standard**," Signal Processing: Image Communication, Vol. 17, No. 1, pp. 3-48, Jan. 2002.

[5] D. Taubman et al, "**Embedded Block Coding in JPEG2000**," Signal Processing: Image Communication Vol. 17, No. 1, pp. 49-72, Jan. 2002.

[6] Ordentlich E. et al, "**A Low-Complexity Modeling Approach for Embedded Coding of Wavelet Coefficients**," Proc. IEEE Data Compression Conf. (Snowbird), pp. 408-417, March 1998.

[7] Lian C. Jr. et al, "**Analysis and Architecture Design of Block-Coding Engine for EBCOT in JPEG2000**" IEEE Trans. Circuits and Systems for Video Technology, Vol. 13, No. 3, pp 219-230, March 2003.

[8] A. Said and W. A. Pearlman, "**A New, Fast, and Efficient Image Codec Based On Set Partitioning in Hierarchical Trees**," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, No. 3, pp. 243-250, June 1996.

[9] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "**Efficient, low complexity image coding with a set-partitioning embedded block coder**,"IEEE Trans. Circuits Syst. Video Technol., vol. 14, no. 11, pp. 1219–1235, Nov.2004.

[10] Pearlman W. A., "**Trends of Tree-Based, Set-Partitioning Compression Techniques in Still and Moving Image Systems**," Proceedings Picture Coding Symposium 2001 (PCS-2001), Seoul, Korea, pp. 1-8, April, 2001.

[11] Ranjan, K. et al, "**Listless Block-tree Set Partitioning Algorithm for Very Low Bit-rate Embedded Image Compression**," Elsevier, International Journal of Electronics and Communications, pp 985-995, 2012.

[12] Danyali H. and Mertins A., "**Flexible, Highly scalable, Object-based Wavelet Image Compression Algorithm for Network Applications**," IEE Proc. Visual Image Signal processing, Vol. 151, No. 6, pp. 498-512, Dec. 2004.

[13] MonauwerAlamand Ekram Khan, "**Listless Highly Scalable Set Partitioning in Hierarchical Trees Coding for Transmission of Image over Heterogeneous Networks**", International Journal of Computer Networking, Wireless and Mobile Communications (IJCNWMC), Vol.2, Issue 3 pp. 36-48, Sep 2012.

[14] Ali Kadhim Al-Janabi, "**Low Memory Set-Partitioning in Hierarchical Trees Image Compression Algorithm**", International Journal of Video & Image Processing and Network Security IJVIPNS-IJENS, Vol.13, No.02, pp. 12-18, April 2013.

[15] http://www.cipr.rpi.edu/research/SPIHT