



An efficient and highly scalable listless SPIHT image compression framework

Ali K. Al-Janabi^{a*} • Hassan K. Al-Musawi^a • Yahya J. Harbi^b

^aUniversity of Kufa, Faculty of Engineering,

Department of Electronics and Communication Engineering, Najaf, Iraq

^bUniversity of Kufa, Faculty of Engineering, Department of Electrical Engineering, Najaf, Iraq

Received 10 19 2020; accepted 07 19 2021

Available 04 30 2022

Abstract: The SPIHT is a powerful image compression algorithm. It has reasonable complexity and produces a quality (or rate) scalable bit-stream. Unfortunately, SPIHT fails to explore the multi-resolution nature of the wavelet transform as it doesn't support resolution scalability. Moreover, it requires a huge computer memory with complex memory management because it utilizes lists with a memory of about 2.5 the image size. This paper proposes three related algorithms. The first algorithm modifies SPIHT to reduce its complexity and improve its efficiency, especially at low bit rates. The second algorithm is the main contribution of the paper. It provides a simultaneous solution to the memory and scalability problems of SPIHT. Memory is reduced by utilizing state marker bits of an average size of 2.5 bits per pixel instead of the lists. Resolution scalability is maintained by coding the resolution levels in incremental order. Consequently, the resulting bit-stream can be easily and efficiently decompressed at numerous qualities and resolutions. This feature is very valuable for modern users that have diverse access bandwidths and display capabilities. The third algorithm has slightly lower complexity and memory than the second algorithm but has slightly lower performance. Another important attribute of our algorithms is that they have a very little increment in complexity in comparison to the original SPIHT algorithm. In contrast, the existing solutions have much more complexity and memory resources.

Keywords: DWT, Quality Scalability, Rate Scalability, Resolution Scalability, Scalable Image Compression, SPIHT

*Corresponding author.

E-mail address: alik.aljanabi@uokufa.edu.iq (Ali K. Al-Janabi).

Peer Review under the responsibility of Universidad Nacional Autónoma de México.

1. Introduction

Transmitting compressed images over heterogeneous networks such as the Internet requires that the compression algorithm be quality and resolution scalable. A quality scalable image compression system creates a bit-stream in which the data is arranged in such a way that the bits that have the higher quality improvement are placed before the other bits. As such, the image can be decompressed at any bit rate while the (full-size) image quality is the best at this rate. In this way, a low bandwidth user can reconstruct a low-quality image while a user that has high bandwidth may reconstruct the image at high quality. On the other hand, a resolution scalable compression creates a bit-stream which consists of several subsets. The first subset contains the data that belong to the lowest resolution (size) image. The next subset (and all the next subsets) contains the necessary data that is required to reconstruct the image at a larger resolution. Therefore, if only the first subset is received, then the image can be reconstructed at the lowest size. If larger image size is desired, the decoder utilizes this subset and the next received subsets. Resolution scalability is useful for users that have diverse display resolutions, such as smartphones, tablets, laptops, desktop computers, and TV. Resolution scalability is also useful for image browsing. For image browsing, the user first receives a fingernail image (a small and rough approximation of the original image) that is ample for deciding if the image must be received fully or not. Just in case of needing the total size image, the user downloads the rest needed information. It is advantageous to combine both types of scalabilities to create a highly or full scalable bit-stream which is both resolution and quality scalable (Al-Janabi, 2019; Rűefenacht et al., 2019).

Scalability imposes an important restriction on the encoder. Specifically, it must operate without any earlier information about the quality and the size at which the image will be recovered by the users. Thus, the coder has to compress the image at full resolution and quality. After that, the compressed bit-stream is stored on a server. Users with various image quality and display resolution requirements send requests to the server, which delivers the appropriate scaled bit-stream to each one of them (Cappellari et al., 2011).

It is well known that the 2-dimensional discrete wavelet transform (2D-DWT) is an indispensable tool for the scalable compression of images. Its local spatiality feature simplifies quality scalability while its multi-resolution nature easily provides resolution scalability. Briefly, the forward 2D-DWT starts by decomposing the original image into four subbands referred to as LL_1 , HL_1 , LH_1 , and HH_1 . Next, the LL_1 subband is also decomposed into another four subbands referred to as LL_2 , HL_2 , LH_2 , and HH_2 . The LL_x subband may be decomposed M times resulting in $3M+1$ subbands. At any decomposition stage, the LL_m subband, $m = 1, 2 \dots M$, represents a good approximation copy of the original image at reduced resolution with a size equal to $1/2^{2m}$ the original image size.

The inverse 2D-DWT starts by combining the LL_M subband with the HL_M , LH_M , and HH_M subbands to reconstruct the LL_{M-1} subband. The inverse process can be done up to M times to obtain a replica of the original image. Figure 1 shows the effect of the 2D-DWT on the test image "Camera Man" for two decomposition levels ($M = 2$). The first decomposition level splits the image into LL_1 , HL_1 , LH_1 , and HH_1 subbands and the second level splits the LL_1 subband into LL_2 , HL_2 , LH_2 , and HH_2 subbands. As seen, the LL_1 and the

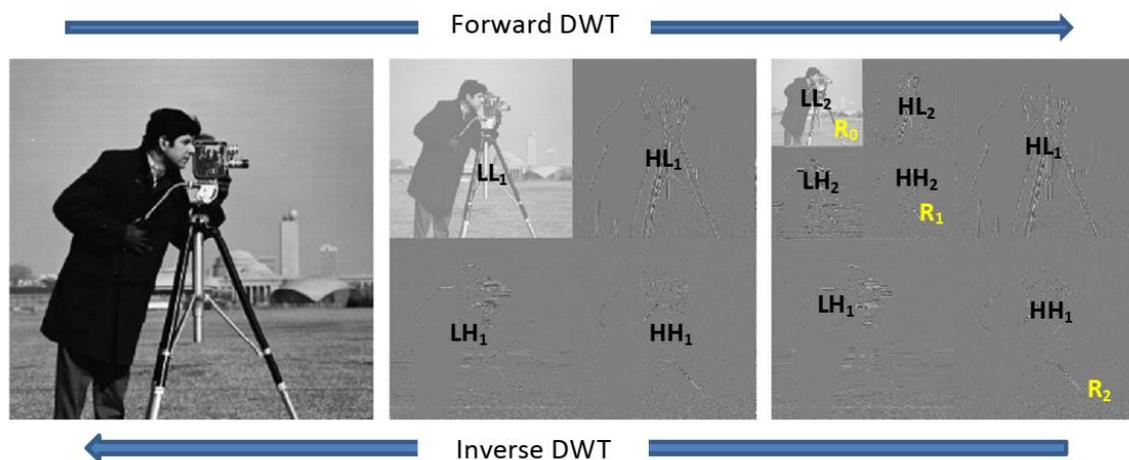


Figure 1. The effect of two levels of 2D-DWT on the "Camera Man".

LL_2 subbands are good approximated copies of the original image with sizes equal to $1/2^2 = 1/4$ and to $1/2^4 = 1/16$ the original image size respectively (Van Fleet, 2019; Vetterli, 2001).

The wavelet subbands are arranged into $M+1$ resolution levels labeled $R_0, R_1 \dots R_M$. The lowermost resolution level, R_0 , consists of the LL_M subband only. Each one of the next levels R_m , $1 \leq m \leq M$, consists of the three subbands HL_{M-m+1} , LH_{M-m+1} , and HH_{M-m+1} that are necessary to reconstruct the LL_{M-m} subband during the inverse 2D-DWT process. Refer to Figure 1, the wavelet image has three resolution levels (R_0, R_1 , and R_2) that are represented by the yellow colour located at the down-right corner boundary of the corresponding resolution level. At the decoder, an image at the lowest resolution (1/16 the image size) can be obtained directly from the LL_2 subband. The image may be recovered at higher resolution by combining the LL_2 subband with the three subbands of R_1 (HL_2, LH_2 , and HH_2), and performing one stage of inverse 2D-DWT to obtain LL_1 (1/4 the image size). Finally, the image may be recovered at the biggest resolution (the entire image size) by combining the LL_1 subband with the 3 subbands of R_2 (HL_1, LH_1 , and HH_1), and carry out another stage of inverse 2D-DWT to get a replica of the original image. Therefore, a resolution scalable bit-stream can be easily achieved if the resolution levels are encoded successively and are identifiable within the compressed bit-stream (Taubman et al., 2002).

The set partitioning in hierarchical trees (SPIHT) (Said & Pearlman, 1996) is a powerful wavelet-based image compression algorithm. Its main pros are it has reasonable complexity, good PSNR (peak signal to noise ratio) vs. the bit rate performance, and it creates a quality scalable bit-stream. At its invention time, the last feature was very interesting as it adds freedom to the user to select the desired quality of the received image very easily. However, this feature is not sufficient nowadays for users that have devices with diverse display resolution capabilities such as laptops smartphones, tablet PCs, etc. As such, adding resolution scalability to SPIHT to generate a highly (rate and resolution) scalable bit-stream is very valuable (Cappellari et al., 2011; Wu et al., 2013). The second important weakness of SPIHT is its massive memory consumption, and complex memory management, which are caused by the use of three linked lists to save the image pixels' coordinates. These lists consume memory about 2.5 times the DWT image (Chew et al., 2009; Singh & Butola, 2015) which represents a true constraint for low memory devices such as wireless sensors (Deepthi et al., 2018; ZainEldin et al., 2015) or for compressing volumetric medical images (Kamargaonkar & Sharma, 2016; Panjavamam & Bhuvanewari, 2017). In addition, in the case of compressing multi-components images such as RGB colour images in parallel, the algorithm would need a memory of about $3 \times 2.5 = 7.5$ times the DWT image which represents a very serious obstacle. Finally, from the scalability point of view, using the

linked lists prohibits resolution scalability due to the recurrent process of removing/adding pixels from/to lists, which makes the stored pixels in these lists not ordered according to the resolution levels (Cappellari et al., 2011; Taubman et al., 2002).

The SPIHT starts by applying the octave 2D-DWT to the image for M decomposition levels (normally $M = 5$). Next, every DWT coefficient c_{ij} is quantized to $[c_{ij}]$, where $[x]$ is the closest integer $\leq x$. Then, every $[c_{ij}]$ is represented using K bits (typically 16 bits), where the first bit is the sign bit (e.g., 1 for negative and 0 for positive coefficient), and the remaining $K - 1$ bits are the magnitude bits. SPIHT creates a quality scalable bit-stream using a power of two threshold coding combined with bit-plane coding by which the DWT coefficients are coded on a per-bit basis starting from the first non-zero MSB (most significant bit) to the LSB (least significant bit). The first non-zero MSB is determined if the threshold TH is selected to be:

$$TH = 2^{\lfloor \log_2 |c_{\max}| \rfloor} \quad (1)$$

where c_{\max} is the magnitude of the pixel that has the maximum value in the DWT image, and $|x|$ is the absolute value of x . The next bit(s) is encoded by decremented TH to $TH/2$ until TH equals 1. TH is sent to the decoder to identify the first non-zero MSB. At each bit-plane coding pass, a DWT coefficient equals to or greater than TH is deemed significant (SG); otherwise, it is deemed insignificant (ISG). Similarly, a set consisting of several coefficients is deemed SG if it contains one or more SG coefficients.

A closer look at Figure 1 reveals that there exists a shadow for the DWT image in all subbands. This means that if a coefficient located at a given position in the LL_M subband is ISG, then the coefficients located at the other subbands that have the same position (with respect to their subbands), are also expected to be ISG. SPIHT exploits this feature to collect as many ISG coefficients as possible and coding them by one symbol. This is attained by combining these ISG coefficients together to build trees referred to as spatial orientation trees (SOTs). The primary roots of these SOTs are the pixels of the LL_M subband excluding the top-left pixel in every set of (2×2) pixels. Every one of the three roots in each (2×2) set is considered a parent for four offspring located at HL_M, LH_M , and HH_M respectively according to its orientation. That is, the top-right pixel is linked to HL_M , the down-left pixel is linked to LH_M , and the down-right pixel is linked to HH_M . Then every pixel in subbands HL_m, LH_m , and $HH_m, M \geq m \geq 2$, is considered a root (parent) to four offspring located at HL_{m-1}, LH_{m-1} , and HH_{m-1} respectively. Notice that the pixels in the HL_1, LH_1 , and HH_1 subbands have no offspring as they are the leaves of the trees. Figure 2 describes the SOTs of the first set of size (2×2) pixels in the subband LL_2 for an image decomposed to two 2D-DWT levels, where each numbered

colour represents a root of an entire SOT that extends across several resolution levels, and the unnumbered colour represent the leaves of these SOTs.

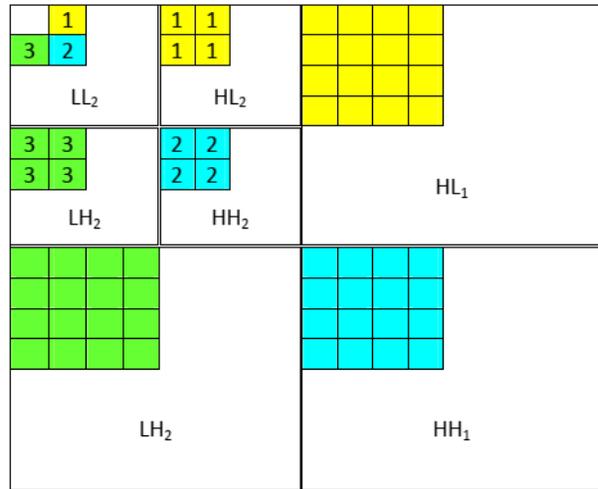


Figure 2. The spatial orientation trees for a 2D-DWT image with two decomposition levels.

SPIHT utilizes three linked lists referred to as the list of insignificant pixels (LIP), the list of insignificant sets (LIS), and the list of significant pixels (LSP). The LIP and LSP keep the (i, j) coordinates of the ISG, and SG pixels respectively while the LIS keeps the (i, j) coordinates of the roots of the SOTs. SPIHT makes use of two type of roots. A root of type A represents the parent of a whole SOT, while a root of type B represents the parent of a partial SOT, which is a whole SOT excluding its four direct offspring (children). The LSP is initialized empty, the LIP is initialized by the (i, j) coordinates of the pixels in the LL_M subband, and the LIS is initialized by the (i, j) coordinates of the primary parent roots of SOTs, which are the pixels in the LL_M subband excluding the top-left pixel in every (2×2) set.

The first coding pass consists of the sorting sub-pass while the next coding passes consist of the sorting and the refinement sub-passes. Over the sorting sub-pass, every pixel c_{ij} in LIP is examined for the current threshold TH. If c_{ij} is ISG, a 0 is sent to the output bit-stream. If c_{ij} is SG, a 1 and its sign bit are sent to the bit-stream, and the pixel's coordinates are transferred to LSP for refinement in the next passes. Next, each root r_{ij} in LIS is examined. If r_{ij} is of type A, its entire SOT is constructed and examined. If the SOT is still ISG, a 0 is sent to the bit-stream. Conversely, if the SOT is SG, then a 1 is sent to the bit-stream, and the four direct offspring of r_{ij} are examined. If an offspring is ISG, a 0 is sent to the bit-stream and its (i, j) coordinates are added to LIP to be tested in the next coding passes. If the offspring is SG, a 1, and its sign bit are sent to the bit-stream, and its (i, j) coordinates are added to LSP for

refinement during the next passes. Finally, if r_{ij} is of type A and has grandchildren (i.e., it lies in LL_M to HH_2 subbands), then r_{ij} is removed from current position and added to the end of LIS as a type B root to be examined later on at the same pass. Alternatively, if r_{ij} is of type B, so a partial SOT that excludes its direct four offspring is constructed and examined again. If this partial SOT is SG, then a 1 is sent to the bit-stream, r_{ij} is removed from LIS, and each one of its four offspring is added to the end of LIS as a type A root to be examined later on at the same pass. If the partial SOT is ISG, then a 0 is sent to the bit-stream and r_{ij} is kept in LIS to be examined again in the following coding pass.

All pixels in LSP are refined during the refinement sub-pass, with the exception of those added during the current pass. A pixel c_{ij} is refined by sending its n_{th} MSB to the bit-stream. After finishing the sorting and refinement sub-passes, the threshold TH is halved to begin a new coding pass. This process is repeated till the threshold is equal to 1, which means that all the pixels are encoded.

Several works interested in increasing the PSNR vs. the bit rate performance of SPIHT are presented cf. (Çekli & Akman, 2017; Cekli & Akman, 2019; Lee & Hung, 2018; Li & Liu, 2017; Sri & Sahu, 2019). The modified SPIHT (MSPIHT) algorithm presented by Rema et al. (2015) is one of the most efficient algorithms found in the literature. The MSPIHT algorithm used modified SOTs that ensure that the algorithm codes the more significant information in the initial bit-planes. It enhanced the PSNR at very low bit rates (less than 0.5 bpp) but gave approximately the same PSNR for bit rates greater than 0.5 bpp. Furthermore, MSPIHT has the same limitations as SPIHT regarding memory consumption and management. Al-Janabi (2013) proposed an efficient, low complexity, and low memory algorithm termed the Single List SPIHT (SLS). It reduced the memory consumption of SPIHT to about 80% by using one list of size equal to 1/4 the image size and two-state marker bits per pixel. At the same time, it preserved the PSNR and the low complexity features of SPIHT.

Danyali and Mertins (2004) focused on solving the scalability problem of SPIHT in their highly scalable-SPIHT (HS-SPIHT) algorithm. The HS-SPIHT algorithm added resolution scalability to SPIHT by utilizing resolution-dependent sorting passes with the related resolution-dependent linked lists. That is, for every resolution level R_m , $m = 0, 1, \dots, M$, a set of LIP, LSP, and LIS lists are employed. Therefore, there are LIP_m , LSP_m , and LIS_m . In each coding pass, the coder starts encoding from resolution level 0 and proceeds to the utmost level M. The algorithm first does the sorting sub-pass for the coefficients within the LIP_m , and then it processes the roots in the LIS_m in the same way as done in SPIHT. In the refinement sub-pass, each pixel in LSP_m is refined by sending its n_{th} MSB to the bit-stream. After finishing the sorting and refinement sub-passes for resolution R_m , the

process is repeated for the lists related to the subsequent resolution levels. Unfortunately, the HS-SPIHT algorithm has also the same weakness of SPIHT regarding memory consumption. Furthermore, it must deal with 3M linked lists instead of 3. This needs extra efforts for memory management and consequently increases the complexity of the algorithm as compared to the SPIHT algorithm.

Alam and Khan (2012) introduced the Listless HS-SPIHT (LHS-SPIHT) algorithm. It utilized the linear indexing technique to convert the DWT image from a 2D array to a 1D array of the same size to simplify the tracking of set partitioning. The functionalities of the lists LIP_m , LSP_m , and LIS_m of HS-SPIHT are performed using fixed-size state marker bits. That is, rather than adding the ISG pixels to LIP_m , the SG pixels to LSP_m , and the roots to LIS_m , the corresponding state marker bit is updated accordingly. For example, updating the marker bit of a pixel c_{ij} that lies in R_m to 1 is equivalent to adding its coordinates to LIP_m . Consequently, rather than examining the pixels in every LIP_m only, the algorithm must examine the state marker bit of every pixel within the DWT image and codes the pixels that have a marker bit equals to 1 only. The same thing occurs for the SG pixels in every LSP_m and the roots in every LIS_m . This means that LHS-SPIHT must examine all the image pixels twice, and must examine the roots in subbands LL_M to HH_2 once in each coding pass. As a result, the algorithm's complexity rises when compared to HS-SPIHT, and consequently the complexity rises even more when compared to SPIHT. The average memory of the state marker is 4 bits per pixel. Since each pixel in the DWT image is represented by 16 bits, so the marker bits need additional memory equal to 1/4 the size of the DWT image. More importantly, using the linear indexing technique necessitates either saving the 2D-DWT image into the main memory and then writing it into a 1D array or both 2D-DWT images and the 1D array must be available in memory at the same time. Unfortunately, the former solution is time-consuming while the latter one demands additional memory equals to the DWT image (Drozdek, 2012). So, the actual memory of LHS-SPIHT is equal to $1+1/4 = 1.25$ the image size.

2. The proposed algorithms

This section first presents a modified version of the SPIHT algorithm termed the simplified SPIHT (SSPIHT) as it employs one type of roots instead of two. Then, the proposed Highly Scalable Listless SPIHT (HLSL) algorithm that represents the current work's main contribution is introduced. The proposed work reduces the memory of the SLS algorithm presented in (Al-Janabi, 2013) further by replacing the list with state marker bits that consume on average 0.5 bit per pixel. More importantly, it upgrades the SLS algorithm to create a highly

scalable bit-stream that can be decompressed at numerous bit rates and resolutions by processing the resolution levels in each coding pass incrementally. Finally, the Interleaved HLSL (IHLSL) which is a simplified version of HLSL is put forward.

2.1. The SSPIHT algorithm

Recall that in SPIHT, there are two types of roots. A root of type A is coded by constructing its complete SOT, and if this SOT is SG, and the root has grandchildren, then the root type is changed to type B root and added to LIS again. A type B root is coded by constructing its partial SOT (which excludes the four direct offspring of the root). And if this SOT is SG, then the root is taken out from LIS and its four direct offspring are added to LIS as type A roots. This means that for every root, the SOT must be constructed twice in each coding pass; one when the root is of type A, and one when it is transformed to type B. The proposed SSPIHT doesn't need to use two types of roots. It is based on the observation that if a complete SOT of a given root is SG, then its partial SOT has a high probability of being SG too (Wenjun et al., 2002). Thus, the condition of adding its four direct offspring to LIS as roots of type A has a high chance of being satisfied. In other words, there is no need to do the condition test.

In SSPIHT, the coding of the LIP and LSP is done exactly as in SPIHT. However, the LIS is processed differently as follows: for every root in LIS, its whole SOT is constructed and examined for significance. If the SOT is ISG, a 0 is sent to the bit-stream as before. If the SOT is SG, then a 1 is sent to the bit-stream, and each one of the root's four direct offspring is coded as done in SPIHT. Finally (and more importantly), if this SG root has grandchildren (i.e., it lies in LL_M to HH_2 subbands), then the root is taken out from LIS and each one of its four direct offspring is added as a root to the end of LIS to be examined later during the present pass. This modification avoids the extra processing time for constructing the partial SOT for every root of type B and eliminates the need of using the type bit. In addition, SSPIHT has superior PSNR performance than the MSPIHT algorithm presented by Rema et al. (2015) which is optimized to improve the PSNR at low compression bit rates as will be shown experimentally in the next section.

2.2. The HLSL algorithm

The proposed HLSL algorithm benefits from the multi-resolution characteristics of the 2D-DWT to generate a highly scalable bit-stream that can be decompressed at more than one rate (quality) and more than one resolution (size). Like the LHS-SPIHT algorithm (Alam & Khan, 2012), the proposed HLSL algorithm also makes use of state marker bits instead of the linked lists used by SPIHT. However, the HLSL has the following advantages over LHS-SPIHT:

- It doesn't use the linear indexing technique to avoid the complexity or the memory increment of this technique as clarified previously.

- HSLs utilizes one marker bit termed δ of size 2 bits for every pixel in the DWT image, and two markers termed α and β of size 1 bit each for every root in the subbands LL_M to HH_2 . Since the total size of these subbands is equal to 1/4 the size of the image, so every root marker needs on average 1/4 bit. So, the total memory consumed by our HSLs algorithm is $2+1/4+1/4 = 2.5$ bits per pixel only. In contrast, LHS-SPIHT used an average of 4 bits per pixel (Alam & Khan, 2012).

- Like SSPIHT, it employs one type of roots.

- It eliminates the need to examine all the image pixels twice per pass. The main idea behind this is based on the fact that the pixels stored in the LIP represent the ISG offspring, and the pixels stored in the LSP represent the SG offspring of the roots of the SOTs that are found SG. So, these ISG and SG pixels can be easily deduced from the parent SG roots. In this way, the LIP and LSP can be dispensed by only computing the offspring that belong to the roots of the SG SOTs in each coding pass. Since these roots are located in subbands LL_M to HH_2 which have a size equal to 1/4 the DWT image size, and since only the SG SOTs are processed, then at most only 1/4 the image pixels need to be processed instead of testing all the image pixels two times in each pass as done in LHS- SPIHT (Alam & Khan, 2012). Evidently, this will reduce the algorithm's computational time.

The purpose of the state marker bits δ , α , and β is to indicate the significance status of the corresponding pixel and root as follows:

- $\delta_{ij} = 0$: the pixel c_{ij} is ISG.
- $\delta_{ij} = 1$: c_{ij} has just become SG in the current coding pass.
- $\delta_{ij} = 2$: c_{ij} is found SG in one of the previous coding passes.
- $\alpha_{ij} = 0$: the root r_{ij} has an ISG SOT.
- $\alpha_{ij} = 1$: r_{ij} has a SG SOT.
- $\beta_{ij} = 0$: r_{ij} is not considered for testing in the current coding pass or it is tested in a previous coding pass.
- $\beta_{ij} = 1$: r_{ij} is considered for testing in the current coding passes.

At initialization, HSLs computes the threshold TH using Eq. 1. Then, it sets the marker bit δ of every pixel in the DWT image to 0 to indicate that all the pixels are still ISG, sets the marker bit α for every root to 0 to indicate that all the roots are still ISG, and finally sets the marker bit β of every root in the LL_M subband to 1, and all other roots to 0. Setting the marker bit β_{ij} of the root r_{ij} to 1 is equivalent to adding the (i, j) coordinates of r_{ij} to LIS in SPIHT.

The first bit-plane coding pass consists of the sorting sub-pass only while the other passes consist of the sorting and refinement sub-passes. To preserve resolution scalability in each coding pass, the algorithm scans the DWT image from the lowest resolution level R_0 , which contains the LL_M

subband only, to the utmost resolution level R_M , which contains the HL_1 , LH_1 , and HH_1 subbands. That is, in each coding pass, the algorithm performs the sorting sub-pass for all resolutions incrementally, and then performs the refinement sub-pass for all resolutions incrementally too.

The sorting sub-pass starts by coding each pixel c_{ij} in resolution R_0 by the `code_pixel(c_{ij})` procedure described next. Then, it will proceed to the next resolutions ($R_1 - R_M$). For every resolution R_m , $1 \leq m \leq M$, all the parent roots are examined. It is worth noting that the parent roots of R_m lie in R_{m-1} (i.e., the parent roots of R_1 lie in R_0 and so on). So, in fact, the parent roots are located in ($R_0 - R_{M-1}$). So, for every resolution, R_m , $0 \leq m \leq M - 1$, each root r_{ij} that is found SG in a previous coding pass (i.e., with $\alpha_{ij} = 1$) is processed by computing its direct four offspring O_k , $k = 1, 2, 3, 4$. Then, each offspring O_k is coded as a pixel by the `code_pixel(O_k)`. Notice that this step is left out in the first coding pass as all the roots are still ISG. Finally, for every resolution, R_m , $0 \leq m \leq M - 1$, each root r_{ij} that is considered for testing in the current coding pass (i.e., with $\beta_{ij} = 1$), its SOT is constructed and tested for significance. If r_{ij} has still an ISG SOT, a 0 is transmitted to the bit-stream, and r_{ij} will be examined again in the next pass. If r_{ij} has an SG SOT, a 1 is sent to the bit-stream, α_{ij} is updated to 1 (i.e., r_{ij} is marked as a SG root), and β_{ij} is updated back to 0 (this is equivalent to removing the (i, j) coordinates of r_{ij} from LIS). Then, each one of the r_{ij} 's direct four offspring O_k , $k = 1, 2, 3, 4$ is coded as a pixel by the `code_pixel(O_k)`. Finally, the marker bit of each one of its direct four offspring β_{O_k} is updated to 1 in order to be considered for testing as roots at the next resolution in the current pass. However, the last step is not performed for the offspring that lie at the utmost resolution level R_M because they are the leaves of the trees.

The refinement sub-pass firstly sends every pixel c_{ij} in R_0 that is found SG in the previous passes (i.e., with $\delta_{ij} = 2$) to the `refine_pixel(c_{ij})` procedure, described shortly, for refining. Next, for every resolution level R_m , $1 \leq m \leq M$, each one of its SG roots r_{ij} (i.e., with $\alpha_{ij} = 1$) that lie in R_{m-1} , its direct four offspring O_k , $k = 1, 2, 3, 4$ is coded by the `refine_pixel(O_k)` procedure to refine its magnitude. Lastly, the threshold TH is updated to TH/2 to proceed to the next coding pass until the threshold is equal to 1 indicating that all the pixels are encoded.

The `code_pixel(c_{ij})` procedure works as follows: If c_{ij} is yet marked ISG (i.e., $\delta_{ij} = 0$), c_{ij} is tested for significance. If $|c_{ij}| \geq TH$, (i.e., c_{ij} has just become SG), a 1 and the sign bit are transmitted to the bit-stream, δ_{ij} is set to 1 (this is equivalent to adding the (i, j) coordinates of c_{ij} to LSP in SPIHT), and finally, c_{ij} is updated to $c_{ij} - TH$ if it is positive or to $c_{ij} + TH$ if it is negative. Conversely, if $|c_{ij}| < TH$ (i.e., c_{ij} is yet ISG), a 0 is transmitted to the bit-stream. If $\delta_{ij} = 1$ (i.e., c_{ij} is found SG at

the previous pass, δ_{ij} is updated to 2 to distinguish c_{ij} from those pixels that will become SG at the current coding pass.

In the `refine_pixel(cij)` procedure, if $|c_{ij}| \geq TH$, a 1 is transmitted to the bit-stream, and c_{ij} is updated to $c_{ij} - TH$ if $c_{ij} > 0$ or to $c_{ij} + TH$ if $c_{ij} < 0$. Otherwise, a 0 is sent to the bit-stream.

The pseudocodes of the encoder and decoder of the proposed HSLs are shown in Figure 3a and 3b respectively. The following remarks clarify the operation of the algorithm.

- As mentioned previously, the scalable encoder has to encode the image at the full bit rate (i.e., until $TH = 1$). On the other hand, the decoder may stop when the target bit rate is reached.

- As mentioned before, for a highly scalable bit-stream, each resolution level within each coding pass must be recognizable in the bit-stream. This is easily attained since these levels are encoded incrementally in each coding pass, by adding a marker at the beginning of each resolution level, that designates the length of corresponding the level.

- The decoder algorithm executes the same four steps of the encoder. The difference is that the decoder works inversely. That is, when it receives 0/1, this indicates that the corresponding pixel or SOT is ISG/SG respectively. Then, the decoder follows the same route as the encoder. Notice that if a pixel c_{ij} is SG, the decoder recognizes that the magnitude of c_{ij} is in the range $[TH - 2TH]$, so c_{ij} is reconstructed at the midpoint which is equal to $\pm 1.5 \times TH$ based on its sign bit. Then every received refinement bit increases the precision of the pixel by $\pm TH/2$ according to the received bit and the sign of the pixel. For example, assume that at the encoder the initial threshold $TH = 16$ and $c_{ij} = +19$. c_{ij} is SG as $+19 \geq 16$. So, at pass#1, the encoder sends 1 and the sign bit 0, and it updates c_{ij} to $19 - 16 = 3$. At pass#2, the encoder updates TH to $TH/2 = 8$, and sends 0 as $3 < 8$. At pass#3, it updates TH to 4, and sends 0 since $3 < 4$. Finally, at pass#4, it updates TH to 2 and sends 1 since $3 \geq 2$. The decoder first receives $TH = 16$. At pass#1, the received bits are 1 and 0, so it reconstructs c_{ij} to $+1.5 \times TH = +1.5 \times 16 = +24$. At pass#2, TH is updated to 8 and the received bit = 0, so c_{ij} is updated to $+24 - 4 = +20$. At pass#3, $TH = 4$ and the received bit = 0, then c_{ij} is set to $+20 - 2 = +18$. Finally, at pass#4, $TH = 2$ and the received bit = 1, then c_{ij} is updated to $+18 + 1 = +19$ which is equal to the original value.

- The decoder can reconstruct an image at resolution R_m , $0 \leq m \leq M$ by receiving the data that corresponds to R_0 to R_m and bypassing the data of the other resolutions in each coding pass. Then it performs m stages of inverse 2D-DWT only to recover the image with size equal to $1/2^{2m}$ the size of the original image.

2.3. The IHSLs algorithm

The IHSLs algorithm has only one coding pass per bit-plane instead of two. The new coding pass is termed the merging pass. It merges the sorting sub-pass with the refinement sub-pass. However, the `code_pixel(cij)` is modified to code the pixels that are still ISG and to refine the pixels that became SG in the preceding coding passes. The merged `mcode_pixel(cij)` starts by examining the pixel's status bit δ_{ij} . If $\delta_{ij} = 0$ (i.e., c_{ij} is yet untested), then c_{ij} is examined for significance and coded exactly as done in the `code_pixel(cij)`. On the other hand, if $\delta_{ij} = 1$ (i.e., c_{ij} is found SG at a previous pass), then c_{ij} is refined as given in the `refine_pixel(cij)`. Notice that in the adopted coding method, the pixel may be either SG or ISG. Therefore, one marker bit for δ is sufficient: $\delta_{ij} = 0$ if c_{ij} is ISG, and $\delta_{ij} = 1$ if c_{ij} is SG. The pseudocodes of the merged `mcode_pixel(cij)` and the merged `mdecode_pixel(cij)` procedures are shown in Figure 4. The same notes of the decoding procedure explained previously apply to `mdecode_pixel(cij)`.

Compared to the HSLs algorithm, the IHSLs algorithm runs slightly faster because it performs one coding pass per bit-plane instead of two. In addition, it has a slightly lower memory requirement as it uses one bit for δ instead of two. So, the total memory of IHSLs is 1.5 bits per pixel instead of the 2.5 bits per pixel that are required by HSLs. However, the cost of these benefits is a slight decrement in PSNR performance as compared to that of HSLs due to the coding pass merging

3. Results and analysis

The proposed algorithms are implemented using Borland C++ v. 5.02 under Intel μ processor Core i3 PC with 1.8 GHz CPU, and 2 GB RAM. The popular grayscale test images "Lena," "Barbara," "Goldhill," and "Mandrill" of size (512x512) pixels are used in the simulation. As with other algorithms, the image is transformed by the octave 2D-DWT with 5 decomposition levels using the CDF 9/7 wavelet filter (Vetterli, 2001). The PSNR of the algorithm vs. the bit rate, as well as its computational time vs. the bit rate, are used to represent the results. The bit rate is the average number of bits per pixel (bpp) of the compressed image. The PSNR measures how similar the original and recovered images are. It is defined as (Al-Janabi, 2015):

$$PSNR = 10 \log_2 \frac{p_{max}^2}{MSE} \text{ decibels}(dB) \quad (2)$$

where p_{max} is the maximum pixel value in the original image ($p_{max} = 2^8 = 255$) for grayscale images), and MSE is the mean-squared error between the original image I_o and the recovered image I_r , given by:

```

Step 1: Initialization
• TH = 2⌊log2|cmax⌋;
• send(TH) to bitstream;
• ∀ cij ∈ DWT Image, set δij = 0;
• ∀ rij ∈ R0 to RM-1, set αij = 0;
• ∀ rij ∈ LLM, set βij = 1;
Step 2: The sorting sub-pass
• ∀ cij ∈ LLM, do: code_pixel(cij);
• ∀ rij ∈ R0 to RM-1 & αij = 1, do:
  ▪ ∀ Ok ∈ rij, k = 1 to 4, do:
    ✓ code_pixel(Ok);
• ∀ rij ∈ R0 to RM-1 & βij = 1, do:
  ▪ If (SOTrij) is SG do:
    ✓ send(1) to bitstream;
    ✓ βij = 0;
    ✓ ∀ Ok ∈ rij, k = 1 to 4, do:
      ○ code_pixel(Ok);
      ○ βOk = 1;
  ▪ Else, send(0) to bitstream;
Step 3: The refinement sub-pass
• ∀ cij ∈ LLM & δij = 2 do:
  ▪ refine_pixel(cij);
• ∀ rij ∈ R0 to RM-1 & αij = 1, do:
  ▪ ∀ Ok ∈ rij, k = 1 to 4, & δk = 2, do
    ✓ refine_pixel(Ok);
Step 4: Threshold update
• TH = TH/2;
• If(TH > 1), goto step 2;
• Else, end Encoding;
code_pixel(cij){
• If(δij = 0) do:
  ▪ If(|cij| ≥ TH) do:
    ✓ send(1) to bitstream;
    ✓ send(sign bit) to bitstream;
    ✓ δij = 1;
    ✓ If(cij > 0), cij = cij - TH;
    ✓ Elseif(cij < 0), cij = cij + TH;
  ▪ Else, send(0) to bitstream;
• Elseif(δij = 1), δij = 2;
refine_pixel(cij){
  ▪ if(|cij| ≥ TH) do:
    ✓ send(1) to bitstream;
    ✓ If(cij > 0), cij = cij - TH;
    ✓ Elseif(cij < 0), cij = cij + TH
  ▪ Else, send(0) to bitstream; }
(a) The Encoder
    
```

```

Step 1: Initialization
• receive(TH) from bitstream;
• ∀ cij ∈ DWT Image, set δij = 0;
• ∀ rij ∈ R0 to RM-1, set αij = 0;
• ∀ rij ∈ LLM, set βij = 1;
Step 2: The sorting sub-pass
• ∀ cij ∈ LLM, do: decode_pixel(cij);
• ∀ rij ∈ R0 to RM-1 & αij = 1, do:
  ▪ ∀ Ok ∈ rij, k = 1 to 4, do:
    ✓ decode_pixel(Ok);
• ∀ rij ∈ R0 to RM-1 & βij = 1, do:
  ▪ If(received bit = 1), do:
    ✓ βij = 0;
    ✓ ∀ Ok ∈ rij, k = 1 to 4, do:
      ○ decode_pixel(Ok);
      ○ βOk = 1;
Step 3: The refinement sub-pass
• ∀ cij ∈ LLM & δij = 2 do:
  ▪ derefine_pixel(cij);
• ∀ rij ∈ R0 to RM-1 & αij = 1, do:
  ▪ ∀ Ok ∈ rij, k = 1 to 4, & δk = 2, do
    ✓ derefine_pixel(Ok);
Step 4: Threshold update
• TH = TH/2;
• If(No. of received bits < Target No. of bits),
  goto step 2;
• Else, end Decoding;
decode_pixel(cij){
• If(δij = 0) do:
  ▪ If(received bit = 1) do:
    ✓ δij = 1;
    ✓ receive sign bit;
    ✓ If(sign bit = 0), cij = 1.5TH;
    ✓ Elseif(sign bit = 1), cij = -1.5TH;
• Elseif(δij = 1), δij = 2;
derefine_pixel(cij){
  ▪ If(received bit = 1) do:
    ✓ If(cij > 0), cij = cij + 0.5TH;
    ✓ Elseif(cij < 0), cij = cij - 0.5TH
  ▪ Elseif(received bit = 0) do:
    ✓ If(cij > 0), cij = cij - 0.5TH;
    ✓ Elseif(cij < 0), cij = cij + 0.5TH; }
b) The Decoder
    
```

Figure 3. The pseudocodes of the proposed HSLs Algorithm.

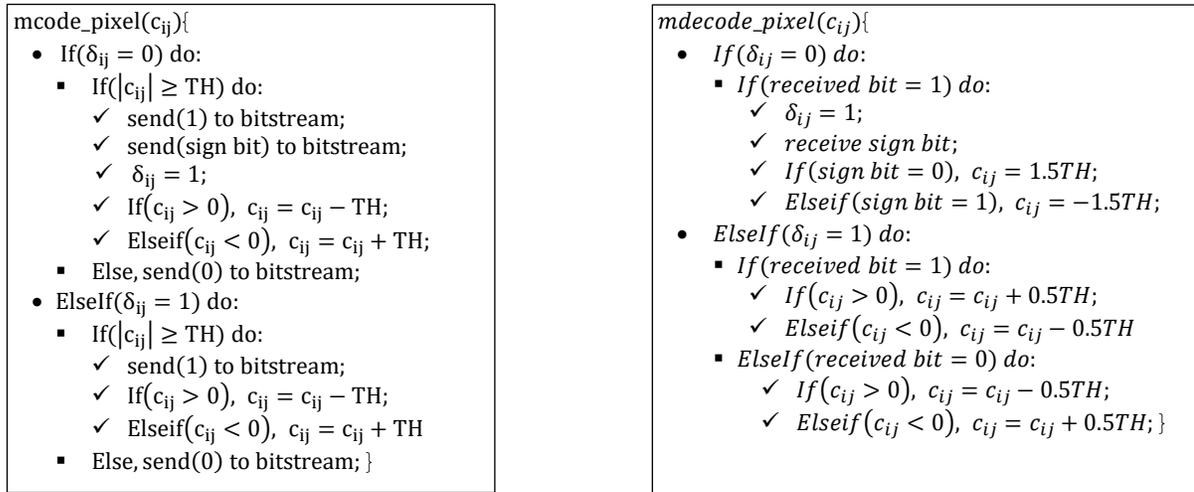


Figure 4. The pseudocodes of the merged `mcode_pixel(cij)` and `mdecode_pixel(cij)`.

$$MSE = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N [I_0(i, j) - I_r(i, j)]^2 \quad (3)$$

where MN is the image size (the number of image pixels). Obviously, for any bit rate, a higher PSNR value is preferred.

3.1. Performance of the proposed SSPIHT algorithm

Table 1 depicts the PSNR vs. bit rate for the proposed SSPIHT, the original SPIHT, and the MSPIHT algorithm (Rema et al., 2015) which is optimized to improve the PSNR at low bit rates. The PSNR of our SSPIHT algorithm is boldfaced where it is higher than that of MSPIHT. As it is clearly shown, the PSNR of SSPIHT is higher for nearly all images at all bit rates. This means that our SSPIHT performs better than SPIHT and MSPIHT. In addition, as will be seen in the next section, the proposed SSPIHT runs faster than the original SPIHT, and therefore it runs also faster than MSPIHT since the latter two algorithms have nearly the same complexity (Rema et al., 2015). Figure 5 shows the original “Lena” image and the decoded ones at several bit rates using SSPIHT

3.2. Performance results of the highly scalable algorithms with full resolution images

Tables 2 and 3 give a comparison of the PSNR vs. the bit rate of the proposed HSLs and IHSLs algorithms with the highly scalable algorithms HS-SPIHT (Danyali & Mertins, 2004) and LHS-SPIHT (Alam & Khan, 2012) at full resolution (size) image. Firstly, it should be noted that the PSNR of HS-SPIHT is higher than that of LHS-SPIHT and the proposed algorithms. This is normal due to removing the lists from LHS-SPIHT and from our algorithms. In other words, the slight PSNR decrement is the price of the huge reduction in computer memory resources.

Secondly, the comparison will be between the LHS-SPIHT and the proposed HSLs and IHSLs algorithms since they don’t use lists. The PSNR of our HSLs and IHSLs algorithms is boldfaced where it is higher than that of LHS-SPIHT. As it can be seen, for the proposed HSLs, the PSNR is mostly higher for nearly all images at all bit rates. This means that the proposed HSLs is on par with the other algorithms in terms of performance. The advantages of the proposed algorithm are its lower memory (2.5 bits per pixel while the LHS-SPIHT needs 4 bits per pixel) as depicted previously, and its lower computational time as demonstrated within the next section. Finally, the PSNR of the IHSLs is slightly lower than that of the HSLs. This result is expected as a price for its speed enhancement due to using of one pass per bit-plane instead of two, and as a cost for its memory reduction (1.5 bits per pixel instead of 2.5 bits per pixel). However, it’s also competitive with the other algorithms.

3.3. Computational time calculation

Table 4 shows the encoding and decoding times of the algorithms measured in milliseconds (msec) vs. bit rate for the “Lena” image at full resolution.

The following observations can be deduced:

- For any algorithm, its decoding time is shorter than its encoding time. This is normal for set partitioning algorithms as the decoder does not need to scan and process the pixels and sets to see if they are SG or not.
- The proposed SSPIHT runs faster than the original SPIHT for all bit rates. This is due to eliminating the necessity to construct the SOTs twice in each coding pass.
- The LHS-SPIHT, which creates a highly scalable bit-stream, runs about two times slower than the SPIHT since it needs to scan and examine all the image pixels two times per coding pass.

Table 1. PSNR vs. bit rate for the proposed SSPIHT, SPIHT, and MSPiHT algorithms.

Bit rate (bpp)	PSNR (dB)								
	Lena 512x512			Goldhill 512x512			Barbara 512x512		
	SPIHT	MSPiHT	Proposed SSPIHT	SPIHT	MSPiHT	Proposed SSPIHT	SPIHT	MSPiHT	Proposed SSPIHT
0.01	11.88	15.31	21.02	11.32	16.32	21.87	11.56	15.29	19.89
0.03	22.31	23.69	23.44	21.99	23.07	23.65	20.34	21.15	21.58
0.05	25.03	25.70	26.49	24.17	24.66	25.60	21.92	22.18	22.88
0.1	28.44	28.68	28.88	26.30	26.51	27.13	23.39	23.55	24.24
0.2	31.74	31.88	31.94	28.40	28.50	29.16	25.64	25.83	26.34
0.5	36.46	36.52	36.27	31.47	31.52	32.30	30.38	30.46	31.11



a) Original "Lena" image



b) "Lena" decoded @ 0.05 bpp, PSNR = 26.49 dB



c) "Lena" decoded @ 0.2 bpp, PSNR = 31.94 dB



d) "Lena" decoded @ 0.5 bpp, PSNR = 36.27 dB

Figure 5. Original "Lena" image and decoded at several bit rates using the proposed SSPIHT algorithm.

Table 2. PSNR vs. bit rate for Lena and Barbara images.

Bit rate (bpp)	PSNR (dB)							
	Lena 512x512				Barbara 512x512			
	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs
0.0625	27.52	26.85	27.35	27.28	22.97	22.59	23.37	23.30
0.125	30.31	29.93	30.04	29.86	24.12	23.65	24.26	24.30
0.25	33.33	33.19	33.00	32.83	26.68	26.75	27.31	27.01
0.5	36.57	36.49	36.24	36.08	30.53	30.48	31.05	30.83
1	39.93	39.58	39.58	39.37	35.28	35.19	36.23	35.77

Table 3. PSNR vs. bit rate for Goldhill and Mandrill images.

Bit rate (bpp)	PSNR (dB)							
	Goldhill 512x512				Mandrill 512x512			
	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs
0.0625	26.27	26.26	26.15	26.11	20.19	20.38	20.26	20.28
0.125	28.02	27.50	27.80	27.70	21.45	21.25	21.27	21.26
0.25	30.19	29.39	29.73	29.80	22.86	22.66	22.58	22.60
0.5	32.40	32.10	32.05	32.13	24.19	24.60	24.68	24.64
1	35.67	35.54	35.40	35.34	28.50	28.30	28.30	28.00

Table 4. The encoding and decoding times vs. bit rate for Lena image.

Bit rate (bpp)	Encoding Time (msec)					Decoding Time (msec)				
	SPIHT	SSPIHT	LHS-SPIHT	HSLs	IHSLs	SPIHT	SSPIHT	LHS-SPIHT	HSLs	IHSLs
0.125	15	12	24	16	14	5	5	11	7	5
0.25	20	16	28	22	18	10	10	17	12	10
0.5	26	23	33	26	24	20	20	28	23	21
1	31	28	38	31	28	25	25	33	27	26
2	63	60	52	64	60	32	32	45	35	33

•The proposed HSLs, which also creates a highly scalable bit-stream, has very little increment in the encoding and decoding times as compared to SSPIHT. This is because that the HSLs needs only to separate and identify the resolution levels.

•Additionally, the HSLs, and SPIHT algorithms have approximately the same processing time. This means that our HSLs added resolution scalability, and reduced the memory requirements of SPIHT without paying any noticeable cost. In contrast, the HS-SPIHT increased the memory requirements, and increased the memory management, while the LHS-SPIHT increased the complexity further against SPIHT.

•As expected, the Proposed IHSLs runs faster than HSLs due to pass merging

3.4. Performance results of the highly scalable algorithms at reduced resolution

A highly scalable decoder may reconstruct the image at a resolution less than that of the original image. Eqs. 2 and 3 can't be used to compute the PSNR in this case because the original and recovered images aren't the same size. Therefore, to be able to give numerical results at reduced resolution, the same manner used in (Danyali & Mertins, 2004) which is also used by Alam and Khan, (2012) will be employed. It utilizes the fact that an image with resolution s , $0 \leq s \leq M$, is the subband LL_{M-s} within the DWT image. So, the original LL_{M-s} and the recovered LL_{M-s} subbands are compared in place of the original and recovered images. For this case, p_{max} would become the maximum pixel value in the subband LL_{M-s} . It can

be shown that for a grayscale image (with 8 bpp), p_{max} is equal to $(255 \times 2^{M-s})$ (Danyali & Mertins, 2004). For instance, if an image with (512×512) pixels is decomposed with $M = 5$ wavelet levels, and the image is recovered at resolution $s = 4$, then $LL_{M-s} = LL_{5-4} = LL_1$ represents the recovered image which has (256×256) pixels (1/4 the original image size), and $p_{max} = 255 \times 2^{5-4} = 510$.

Tables 5 and 6 depict the PSNR vs. bit rate for the images at $s = 4$ (1/4 resolution), while Tables 7 and 8 depict the PSNR vs. bit rate for the images at $s = 3$ (1/16 resolution). The bit rates are computed using the number of pixels in the full-size image. The PSNR of our HSLs and IHSLs algorithms is boldfaced where it is higher than that of LHS-SPIHT. It is worth to note that in Tables 7 and 8, the PSNR of our algorithms in the last row represents the PSNR achieved at full bit rate compression (i.e., the maximum achievable PSNR). For instance, for the “Lena” image, the proposed HSLs achieves PSNR = 64.77 dB @0.45 bpp. Evidently, at this full bit rate (0.45 bpp), the PSNR will be

smaller than that of the other algorithms which calculated the PSNR @0.5 bpp. However, any PSNR higher than 50 dB is considered perfect in terms of image quality (Gu et al., 2013). As it can be seen, for “Barbara” and “Mandrill” images (which have more details than “Lena” and “Goldhill” images), the proposed HSLs performs better than the LHS-SPIHT and the HS-SPIHT for all bit rates. For “Lena” and “Goldhill” images, the PSNR of HSLs, LHS-SPIHT, and HS-SPIHT are very comparable. In addition, the proposed IHSLs has also a slightly lower PSNR than that of HSLs due to the adopted simplifications. On the other hand, it is superior to LHS-SPIHT and HS-SPIHT for “Barbara” and “Mandrill” images at all bit rates (except for “Mandrill” @0.5 bpp for $s = 4$, and @0.125 bpp for $s = 3$). Finally, for “Lena” and “Goldhill” images, the IHSLs is also very comparable to the LHS-SPIHT, and HS-SPIHT algorithms. This suggests that our algorithms are very successful for users that need to recover low-resolution images, such as that smartphones, tablets, etc.

Table 5. The PSNR vs. Bit rate for Lena and Barbara images @1/4 resolution ($s = 4$).

Bit rate (bpp)	PSNR (dB)							
	Lena				Barbara			
	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs
0.0625	28.78	27.98	28.45	28.40	25.94	25.48	26.84	26.73
0.125	32.34	31.84	32.14	31.90	28.41	27.93	29.24	29.20
0.25	37.44	37.21	37.01	36.94	32.64	32.42	33.66	33.65
0.5	43.82	43.52	43.35	43.29	39.19	38.97	39.23	39.19
1	53.25	53.17	53.05	52.92	50.12	50.02	50.19	50.17

Table 6. The PSNR vs. Bit rate for Goldhill and Mandrill images @1/4 resolution ($s = 4$).

Bit rate (bpp)	PSNR (dB)							
	Goldhill				Mandrill			
	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs
0.0625	27.79	27.36	27.61	27.59	21.47	21.28	22.74	22.83
0.125	30.81	29.78	30.21	30.03	23.65	23.45	24.73	24.42
0.25	33.86	32.97	32.79	32.87	28.79	28.59	28.87	28.75
0.5	38.81	38.51	38.62	38.59	31.41	31.21	31.58	30.95
1	49.97	49.73	49.77	49.76	39.42	39.23	39.52	39.48

Table 7. The PSNR vs. Bit rate for Lena and Barbara images @1/16 resolution ($s = 3$).

Bit rate (bpp)	PSNR (dB)							
	Lena				Barbara			
	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs
0.0625	32.38	31.86	32.08	32.05	30.83	29.89	31.93	31.93
0.125	40.03	39.53	40.34	40.28	36.55	35.83	36.03	35.75
0.25	50.73	50.30	50.89	50.63	47.34	46.12	46.52	46.28
0.5	70.52	70.51	64.77	64.75	71.16	70.89	63.75	63.70
			@0.45 bpp	@0.45 bpp			@0.46 bpp	@0.46 bpp

Table 8. The PSNR vs. Bit rate for Goldhill and Mandrill images @1/16 resolution (s = 3).

Bit rate (bpp)	PSNR (dB)							
	Goldhill				Mandrill			
	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs	HS-SPIHT	LHS-SPIHT	Proposed HSLs	Proposed IHSLs
0.0625	31.48	30.86	31.33	31.28	25.40	25.21	26.00	26.13
0.125	37.13	36.12	36.87	36.82	31.49	31.31	30.42	30.62
0.25	47.96	46.87	47.05	46.95	40.98	40.82	40.87	40.87
	70.91	70.71	64.80	64.80	60.05	59.97	64.77	64.77
0.5			@0.48 bpp	@0.48 bpp			@0.52 bpp	@0.52 bpp

4. Conclusions

In this paper, we presented a framework of three related scalable image compression algorithms. As demonstrated within the last section, the SSPIHT algorithm has lower complexity and improved performance than that of the original SPIHT. Additionally, at low bit rates, it provided higher PSNR than that of the MSPIHT algorithm, which is optimized for that purpose. The HSLs algorithm solved the dual problems of SPIHT: the scalability and the massive memory resources without paying any noticeable cost concerning its performance and complexity. In contrast, the HS-SPIHT increased the complexity and the memory while the LHS-SPIHT widely increased the complexity as compared to SPIHT. Moreover, our HSLs algorithm performed better than these algorithms when the image is recovered at reduced resolutions. Finally, the IHSLs algorithm is faster and consumes less memory than that of HSLs. As shown, the only price for these additional advantages is the very slight reduction in PSNR.

Conflict of interest

The authors do not have any type of conflict of interest to declare.

Financing

The authors did not receive any sponsorship to carry out the research reported in the present manuscript.

References

- Alam, M., & Khan, E. (2012). Listless Highly scalable set partitioning in hierarchical trees coding for transmission of image over heterogenous networks. *International Journal of Computer Networking, Wireless Mobile Commun.*
- Al-Janabi, A. K. (2013). Low memory set-partitioning in hierarchical trees image compression algorithm. *International Journal of Video & Image Processing and Network Security IJVIPNS-IJENS*, 13(2), 12-18.
- Al-Janabi, J., & Kadhim, A. (2015). Ultrafast and Efficient Scalable Image Compression Algorithm. *Journal of ICT Research & Applications*, 9(3), 214-235. <https://doi.org/10.5614/itbj.ict.res.appl.2015.9.3.2>
- Al-Janabi, A. K. (2019). Efficient and simple scalable image compression algorithms. *Ain Shams Engineering Journal*, 10(3), 463-470. <https://doi.org/10.1016/j.asej.2019.01.008>
- Cappellari, L., Milani, S., Cruz-Reyes, C., & Calvagno, G. (2011). Resolution scalable image coding with reversible cellular automata. *IEEE Transactions on Image Processing*, 20(5), 1461-1468. <https://doi.org/10.1109/TIP.2010.2090531>
- Çekli, S., & Akman, A. (2017). An efficient SPIHT algorithm and system architecture for image compression. In *2017 25th Signal Processing and Communications Applications Conference (SIU)* (pp. 1-4). IEEE. <https://doi.org/10.1109/SIU.2017.7960338>

- Çekli, S., & Akman, A. (2019). Enhanced SPIHT algorithm with pipelined datapath architecture design. *Electrica*, 19(1), 29-36. <https://doi.org/10.26650/electrica.2018.15101>
- Chew, L. W., Ang, L. M., & Seng, K. P. (2009). Reduced memory spiht coding using wavelet transform with post-processing. In *2009 International Conference on Intelligent Human-Machine Systems and Cybernetics* (Vol. 1, pp. 371-374). IEEE. <https://doi.org/10.1109/IHMSC.2009.101>
- Danyali, H., & Mertins, A. (2004). Flexible, highly scalable, object-based wavelet image compression algorithm for network applications. *IEE Proceedings-Vision, Image and Signal Processing*, 151(6), 498-510. <https://doi.org/10.1049/ip-vis:20040734>
- Deepthi, S. A., Rao, E. S., & Prasad, M. G. (2018). Image transmission and compression techniques using SPIHT and EZW in WSN. In *2018 2nd International Conference on Inventive Systems and Control (ICISC)* (pp. 1146-1149). IEEE. <http://dx.doi.org/10.1109/ICISC.2018.8398984>
- Drozdek, A. (2012). *Data Structures and algorithms in C++*. Cengage Learning.
- Gu, K., Zhai, G., Yang, X., Zhang, W., & Liu, M. (2013). Subjective and objective quality assessment for images with contrast change. In *2013 IEEE International Conference on Image Processing* (pp. 383-387). IEEE. <https://doi.org/10.1109/ICIP.2013.6738079>
- Kamargaonkar, C., & Sharma, M. (2016). Hybrid medical image compression method using SPIHT algorithm and Haar wavelet transform. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)* (pp. 897-900). IEEE. <http://dx.doi.org/10.1109/ICEEOT.2016.7754817>
- Lee, R. C., & Hung, K. C. (2018). New modified SPIHT algorithm for data compression system. *Journal of Medical and Biological Engineering*, 39, 18-26. <https://doi.org/10.1007/s40846-018-0384-z>
- Li, M. S., & Liu, Y. (2017). Image Compression Based on an Improved SPIHT Algorithm. In *2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)* (pp. 932-935). IEEE. <https://doi.org/10.1109/ICCSEC.2017.8446929>
- Panjavamam, B., & Bhuvaneswari, P. T. V. (2017). Performance analysis of SPIHT algorithm for biomedical image transmission. In *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)* (pp. 1-5). IEEE. <https://doi.org/10.1109/ICSCN.2017.8085695>
- Rema, N. R., Oommen, B. A., & Mythili, P. (2015). Image compression using SPIHT with modified spatial orientation trees. *Procedia Computer Science*, 46, 1732-1738. <https://doi.org/10.1016/j.procs.2015.02.121>
- Rüfenacht, D., Naman, A. T., Mathew, R., & Taubman, D. (2019). Base-anchored model for highly scalable and accessible compression of multiview imagery. *IEEE Transactions on Image Processing*, 28(7), 3205-3218. <https://doi.org/10.1109/TIP.2019.2894968>
- Said, A., & Pearlman, W. A. (1996). A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on circuits and systems for video technology*, 6(3), 243-250. <https://doi.org/10.1109/76.499834>
- Singh, Y., & Butola, B. S. (2015). Shadow SPIHT with Discarded Lists. In *2015 1st International Conference on Next Generation Computing Technologies (NGCT)* (pp. 941-943). IEEE. <https://doi.org/10.1109/NGCT.2015.7375259>
- Sri, A., & Sahu, S. S. (2019). Improved fractal-SPIHT hybrid image compression algorithm. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (pp. 1-4). IEEE. <https://doi.org/10.1109/ICCCNT45670.2019.8944769>
- Taubman, D., Ordentlich, E., Weinberger, M., & Seroussi, G. (2002). Embedded block coding in JPEG 2000. *Signal Processing: Image Communication*, 17(1), 49-72. [https://doi.org/10.1016/S0923-5965\(01\)00028-5](https://doi.org/10.1016/S0923-5965(01)00028-5)
- Van Fleet, P. J. (2019). Discrete wavelet transformations: An elementary approach with applications. pp.183-229, John Wiley & Sons. <https://doi.org/10.1002/9781119555414>
- Vetterli, M. (2001). Wavelets, approximation, and compression. *IEEE Signal Processing Magazine*, 18(5), 59-73. <https://doi.org/10.1109/9.952805>

Wenjun, J., Zhongwei, J., & Weile, Z. (2002). An embedded still image code algorithm with rate-distortion optimization. In *IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions* (Vol. 1, pp. 1-4). IEEE.

<https://doi.org/10.1109/ICCCAS.2002.1180558>

Wu, D., Zhang, H., Li, X., & Wang, J. (2013). Multiview Video Coding Based on Wavelet Pyramids. In *2013 International Conference on Computational and Information Sciences* (pp. 225-228). IEEE.

<https://doi.org/10.1109/ICCIS.2013.67>

ZainEldin, H., Elhosseini, M. A., & Ali, H. A. (2016). A modified listless strip based SPIHT for wireless multimedia sensor networks. *Computers & Electrical Engineering*, 56, 519-532.

<https://doi.org/10.1016/j.compeleceng.2015.10.001>