Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Scalable Image Compression Algorithms with Small and Fixed- Size Memory

Ali Kadhim Al-Janabi (alik.aljanabi@uokufa.edu.iq)
University of Kufa
Yahya J. Harbi
University of Kufa
Mohammed Falih Hassan
University of Kufa

Research Article

Keywords: DWT, Image Compression, Quality Scalable Image Compression, Resolution Scalable Image Compression, Highly Scalable Image Compression, SLS, SPIHT

Posted Date: September 9th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2012580/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Scalable Image Compression Algorithms with Small and Fixed-Size Memory

Ali Kadhim Al-Janabi*[1], Yahya J. Harbi[2], Mohammed Falih Hassan[1]

[1]University of Kufa, Faculty of Engineering, Department of Electronics and Communication Engineering, Najaf, Iraq
[2]University of Kufa, Faculty of Engineering, Department of Electrical Engineering, Najaf, Iraq

Corresponding author: Ali Kadhim Al-Janabi, alik.aljanabi@uokufa.edu.iq ORCID: 0000-0002-3824-3791
2nd Author: Yahya J. Harbi, yahyaj.harbi@uokufa.edu.iq ORCID: 0000-0001-9497-4612
3rd Author: Mohammed Falih Hassan, mohammedf.aljanabi@uokufa.edu.iq ORCID: 0000-0002-2995-7442

Abstract

The SPIHT algorithm is characterized by low computational complexity, good performance, and the production of an embedded bitstream that can be decoded at several bit-rates with image quality enhancement as more bits are received. However, it suffers from the enormous computer memory consumption due to utilizing linked lists of size of about 2-3 times the image size to save the coordinates of the image pixels and the generated sets. In additions, it does not exploit the multi-resolution feature of the wavelet transform to produce a resolution scalable bitstream by which the image can be decoded at numerous resolutions (sizes). The Single List SPIHT (SLS) algorithm resolved the high memory problem of SPIHT by using only one list of fixed size equals to just 1/4 the image size, and an average of 2.25 bits/pixel. This paper introduces two new algorithms that are based on SLS. The first algorithm modifies SLS to reduce its complexity and improve its performance. The second algorithm, which is the major contribution of the work, upgrades the modified SLS to produce a bitstream that is both quality and resolution scalable (highly scalable). As such, the algorithm is very suitable for the modern heterogeneous nature of the Internet users to satisfy their different capabilities and desires in terms of image quality and resolution.

Keywords: DWT, Image Compression, Quality Scalable Image Compression, Resolution Scalable Image Compression, Highly Scalable Image Compression, SLS, SPIHT.

I. Introduction

The main target of lossy image compression is to reduce the average number of bits per pixel of the compressed image as much as possible, and at the same time to

attempt to reduce the difference between the original and the recovered images as much as possible. The difference is usually measured by the mean squared error (MSE) between these images. Other important factors for an image compression algorithm are its computational complexity, and its memory consumption. So, the true judgment of any algorithm must consider all these factors [1, 2].

A conventional image compression system permits to recover the image at just one bit-rate (quality) and resolution (size). As the current users have diverse capabilities in terms of bandwidths, display resolutions, processing power, and memory, this compression paradigm will not fit all users. Additionally, for browsing the images over the web, the users prefer to make a quick scan for all the searched images and then select the desired one(s). Evidently, there will be time, bandwidth, memory, and processing power losses when using this type of image compression. On the other hand, with a scalable image compression system, the quality or/and the resolution (size) of the recovered image can be controlled in such a way that permits to the end user to decode the image at the desired quality or/and resolution. In a quality scalable image compression (QSIC), only the quality of the recovered image can be controlled. A QSIC can be attained by encoding the image pixels utilizing some form of bit-plane coding, where each pixel is encoded from its most significant bit (MSB) to its least significant bit (LSB). On the other hand, in a resolution scalable image compression (RSIC), only the resolution of the recovered image can be controlled. Lastly, in a highly (or full) scalable image compression (HSIC), both the quality and the resolution of the recovered image can be controlled [3, 4]. Therefore, the HSIC is very appealing for the needs of modern users due to its versatility to decode the image at various qualities and resolutions.

The dyadic 2Dimensional-Discrete Wavelet Transform (2D-DWT) is a basic tool for scalable image compression because it possesses high energy compaction, localization across space and frequency, and multiresolution properties. An M decomposition levels of the dyadic 2D-DWT partitions the image into 3M+1 subbands. The first level decomposes the image into four subbands labeled $LL_1, HL_1, LH_1,$ and HH_1. Each one of the next levels m, $2 \leq m \leq M$, decomposes the LL_{m-1} subband into $LL_m, HL_m, LH_m,$ and HH_m subbands. The LL_m subband at any level represents a good approximation of the original image at lower resolution. Every stage of the inverse dyadic 2D-DWT combines the LL_m, HL_m, LH_m, and HH_m subbands to reproduce the LL_{m-1}, $m = M, M-1 \ldots 1$. The subbands are organized into M+1 resolution levels $R_0, R_1 \ldots R_M$. Each resolution level $R_m, 0 \leq m \leq M$, is responsible to recover the image at a specific size which is equal to $1/2^{2(M-m)}$ the original image. The lowest resolution level R_0 contains the LL_M subband only. Each one of the next levels contains the three HL_{M-m+1}, LH_{M-m+1}, and HH_{M-m+1}

subbands that are compulsory to rebuilt the $\mathrm{LL}_{\mathrm{M-m}}$ subband during the inverse 2D-DWT process [5, 6]. Figure 1 illustrates the forward and inverse 2D-DWT using two decomposition levels (M = 2). The first level decomposes the image into LL_1, HL_1, LH_1, and HH_1 subbands. The second level decomposes the LL_1 subband into LL_2, HL_2, LH_2, and HH_2 subbands. For M = 2, there are three resolution levels (R_0, R_1, and R_2). At the decoder side, an image at the lowestmost resolution with a size equal to $1/2^{2(\mathrm{M-m})} = 1/2^{2(2-0)} = 1/16$ the image size can be gotten directly from the LL_2 subband. The image may be reconstructed at a 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 the LL_1 with size equal to $1/2^{2(2-1)} = 1/4$ the image size. Finally, the image may be reconstructed at the entire size (full-resolution) by combining the LL_1 subband with the three subbands of R_2 (HL_1, LH_1, and HH_1), and carrying out the last stage of the inverse 2D-DWT to obtain a replica of the original image. Thus, a resolution scalable bitstream can be attained easily if the resolution levels are encoded successively and they are identifiable within the compressed bitstream [3].

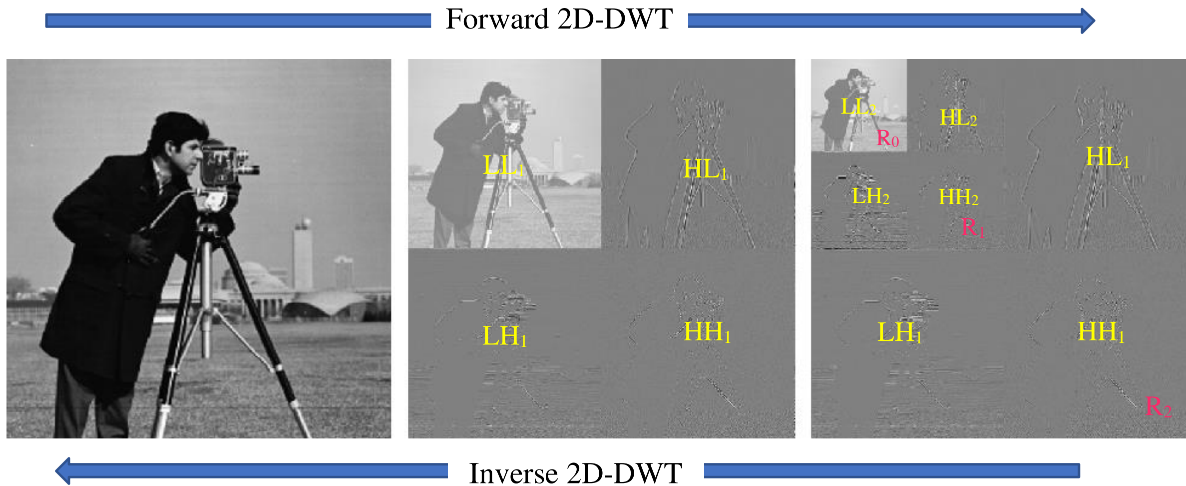

Figure 1: The forward and inverse 2D-DWT with two decomposition levels (from [15])

The set partitioning in hierarchical trees (SPIHT) [7] is one of the benchmarks QSIC algorithms [8, 9]. It has relatively low computational complexity, and has a good MSE vs. bit per pixel performance. In brief, SPIHT first applies the dyadic 2D-DWT to the image. Then, it computes a maximum threshold T based on the maximum wavelet coefficient in the image. In what follows, the wavelet coefficients are called pixels for simplicity. Next, it encodes the image by multiple bit-plane coding passes with halving T ($T = T/2$) in each pass until $T = 1$. A pixel $c(i, j)$ is considered

insignificant (ISG) if $|c(i,j)| < T$, and it becomes significant (SG) when $|c(i,j)| \geq T$. Similarly, a set of pixels is considered ISG if all of its pixels are ISG, and it becomes SG when one or more of its pixels become SG. SPIHT builds trees denoted as the spatial orientation trees (SOTs) by exploiting the correlation between the pixels across the different resolution levels of the dyadic 2D-DWT as follows: the pixels in LL_M subband (level R_0) are grouped into (2×2) pixels. The top-left pixel in each group is excluded (i.e., it does not belong to any SOT). Each one of the other three pixels that is located at (i, j) is considered as a root to four pixels located in LH_M, HL_M, and HH_M subbands respectively (level R_1) according to its orientation. More precisely, the top-right pixel is linked to four pixels in LH_M, the down-left pixel is linked to four pixels in HL_M, and the down-right pixel is linked to four pixels in HH_M. These four pixels are referred to as the offspring O(i, j) of the root located at (i, j). Then, each one of O(i, j) is also in turn considered as a root to four offspring in LH_{M-1}, HL_{M-1}, and HH_{M-1} subbands (level R_2). This recursive linking of the roots continues till the LH_2, HL_2, and HH_2 subbands (level R_{M-1}) is reached. That is, the pixels in LH_1, HL_1, and HH_1 (level R_M) cannot be roots as they are the leaves of tree. Any root at coordinates (i, j) located in levels R_1 to R_{M-1}, the coordinates of its four offspring O(i, j) = {(2i, 2j), (2i+1, 2j), (2i, 2j+1), (2i+1, 2j+1)}, which are located at levels R_2 to R_M respectively. The SPIHT assumes that for a given SOT, if its parent root is ISG, then it is expected that all pixels in the SOT are also ISG. Hence, the complete SOT can be encoded in a single bit.

Unfortunately, SPIHT needs an enormous computer memory to save the coordinates of the pixels, and the roots of the SOTs. More specifically, it employs three linked lists labelled the list of insignificant pixels (LIP), the list of significant pixels (LSP), and the list of insignificant sets (LIS). These lists need a computer memory around 2-3 times the DWT image [10]. In addition, the memory management of the linked lists is complex and time consuming because they must be accessed randomly due to adding/removing elements to/from them continually. Furthermore, the length of each list cannot pre-fixed as the number of ISG and SG pixels, and the ISG SOTs is not known. Therefore, we either use the dynamic memory approach or initialize each list to the maximum size. Regrettably, the first solution is very complex, while the second one increases the memory requirements further [11]. Lastly, the spanning of the SOTs across the different resolution levels together with the random access of the lists, inhibit to exploit the multi-resolution features of the dyadic 2D-DWT for producing a resolution scalable bitstream [3].

Most reduced memory SPIHT algorithms that exist in the literature adopt the linear indexing technique to map the DWT image into a 1D array of the same image size [12-14]. However, this technique demands either storing the DWT image into the

main memory and then writing it into a 1D array or both DWT image and the 1D array must be available in the RAM at the same time [15]. Unfortunately, the former solution is time-consuming while the latter one demands extra memory equals to the DWT image [11]. These constrains prohibit to use this approach for low memory and/or low power processing units such as wireless sensors [16].

A. K. Al-Janabi proposed a reduced memory SPIHT called the Single List SPIHT (SLS) without using the linear indexing technique [17]. It employs one list of fixed size equal to ¼ the DWT image, and an average of 2.25 bits/pixel markers. It was demonstrated that the SLS algorithm kept nearly the same complexity and had better performance than the original SPIHT algorithm, with memory saving of about 75%.

Monauwer et al. [18] mitigated the high memory requirements and the lack of resolution scalability in the SPIHT is his Listless Highly Scalable-SPIHT (LHS-SPIHT) algorithm. It replaced the lists by state marker bits with average memory of 4 bits/pixel. Unfortunately, LHS-SPIHT must test all the pixels and the all roots of the SOTs (which are lied in resolution levels R_0 –R_{M-1}) two times in each bit-plane coding pass. Thus, the complexity of LHS-SPIHT rises significantly compared with the original SPIHT. Equally important, the algorithm also adopts the linear indexing technique. Hence, it has the same cons mentioned above.

In [15], A. K. Al-Janabi et. al., proposed the Highly Scalable Listless SPIHT (HSLS) algorithm. HSLS has the following advantages over LHS-SPIHT:

- It does not rely on the linear indexing technique. Hence, it avoids the complexity or memory increment as clarified above.
- It needs to examine at most 1/4 the image pixels only in each bit-plane coding pass. In contrast, the LHS-SPIHT algorithm must examine all the image pixels twice in each coding pass.
- The total memory of the state marker bits is 2.5 bits per pixel instead of 4.
- HSLS has better performance especially when the image is reconstructed at low resolution.

However, the HSLS algorithm also suffers from some complexity increment and performance decrement due to removing all lists. In this paper, we first introduce the Modified SLS algorithm (MSLS). The modifications reduce the complexity of the SLS algorithm and enhance algorithm’s performance slightly. Then, we present the Highly Scalable-MSLS (HS-MSLS) which is the major contribution of the work. The HS-MSLS upgrades the MSLS to produce a highly scalable bitstream by adding resolution scalability to it. The main feature of the proposed HS-MSLS algorithm is

that the resolution scalability is added without any noticeable complexity increment nor performance decrement as compared to MSLS. In contrast, most of the existing highly scalable algorithms may involve complexity increment and/or performance deterioration in comparison to the rate scalable counterpart algorithms [18, 19].

The rest of the paper is ordered as follows: section II summarizes the SLS algorithm. Section III introduces the proposed MSLS and HS-MSLS algorithms. Section IV provides the simulation results of our work, and other related works for the purpose of comparison. Finally, section V concludes the paper.

II. Overview of the SLS Algorithm

Like other wavelet-based algorithms, the image is first transformed using an M-levels (M = 3 – 6) of the 2D-DWT. The (9, 7) 2D-DWT [6] is usually employed for lossy image compression, while the (5, 3) 2D-DWT [20] is employed for lossless image compression. The (9, 7) transform is more efficient than the (5, 3) transform in terms of energy compaction. However, the (9, 7) transform produces floating-point numbers, while the (5, 3) transform produces integer numbers. So, if the (9, 7) transform is selected, the floating-point coefficients must be quantized to the nearest integers before coding. This is the main source of information loss.

The central idea of SLS is that the pixels stored in LSP and LIP are the four offspring of a root that has a SG SOT. These offspring can therefore be inferred from the SG root itself rather than being kept in these lists. Eliminating these lists will reduce the memory greatly as LIP and LSP occupy about 75% of the total memory. At the first glance, it seems that the need to deduce the four offspring from the parent root leads to complexity increment. However, eliminating these lists reduces the complexity of the algorithm due to reducing the memory management overhead.

The SLS algorithm employs a single list only labelled the List of Root Sets (LRS) that stores the (i, j) coordinates of the roots of the SOTs. As shown, these roots lie in the subbands that belong to R_0–R_{M-1}, i.e., except the subbands that belong to R_M (HL_1, LH_1, and HH_1 subbands). So, the maximum size of LRS is ¼ the image size. Additionally, when a root is added to LRS, it won’t be removed. This coding paradigm enables to implement LRS as an ordered simple 1D array that is sequentially accessed using the first in first out (FIFO) method which is widely known to be the fastest access method [11]. Clearly, by making memory management simpler, the algorithm's complexity will be reduced further.

The function of the LIP and LSP is performed by providing each pixel $c(i,j)$ by 2 bits state marker referred to as $\sigma(i,j)$ to specify the pixel's type as follows:

- $\sigma(i,j) = 0$: $c(i,j)$ is an ISG or untested pixel.
- $\sigma(i,j) = 1$: $c(i,j)$ becomes SG.
- $\sigma(i,j) = 2$: $c(i,j)$ is a visited SG (VSG) pixel that is found SG in one of the previous coding passes.

Each entry in LRS has a one-bit marker termed $\delta(i,j)$ that is initialized to 0 to indicate that the root $r(i,j)$ is ISG and updated to 1 when $r(i,j)$ becomes SG. Since the maximum size LRS is ¼ the image size, so the average memory of the overall marker bits (σ and δ) is 2.25 bits/pixel.

At initialization, SLS first computes the maximum bit-plane (b_{max}) based on the maximum pixel value in the quantized DWT image (W) as follows:

$$b_{max} = \left\lfloor log_2 \{\max_{\forall(i,j)\in W} |c(i,j)|\} \right\rfloor \qquad (1)$$

b_{max} is then sent to the decoder within the bitstream. Next, it sets the threshold T to:

$$T = 2^{b_{max}} \qquad (2)$$

Finally, it saves the (i,j) coordinates of every pixel in LL_M that has offspring in LRS as ISG roots, i.e., $\delta(i,j) = 0$.

After initialization, SLS performs several coding passes. Each coding pass corresponds to a given threshold T, and consists of the sorting and the refinement sub-passes. The sorting sub-pass starts by coding all the pixels in LL_M subband as follows: if $c(i,j)$ is untested or yet ISG ($\sigma(i,j) = 0$)), it is tested for significance. If it becomes SG, then 1, and its sign bit are sent to the bitstream, its marker bit $\sigma(i,j)$ is updated to 1 to indicate that $c(i,j)$ becomes SG. If $c(i,j)$ is still ISG, a 0 is sent to the bitstream. On the other hand, if $c(i,j)$ is found to be SG ($i.e., \sigma(i,j) = 1$)), it is marked as VSG by setting $\sigma(i,j) = 2$, in order to be refined latter on in the current coding pass. This step is necessary to differentiate between these pixels and the pixels that will become SG during the current pass. Next, every root $r(i,j)$ in LRS is tested and coded accordingly. If $r(i,j)$ is yet ISG ($\delta(i,j) = 0$)), its SOT is constructed and its significance is checked with respect to T. If the SOT stills ISG, a 0 is sent to the bitstream. If it becomes SG, a 1 is sent to the bitstream, $\delta(i,j)$ is updated to 1 to indicate that $r(i,j)$ is now SG. Then, each one of its four offspring

O(i, j) is coded as a pixel as given above. Finally, if O(i, j) don't lie in the LH_1, HL_1, and HH_1 subbands, i.e., in the highest resolution level (R_M), O(i, j) are added to LRS as ISG roots be coded in the same manner at the current coding pass. On the other hand, if the $r(i,j)$ is found to be SG in one of the previous passes ($\delta(i,j) = 1$), then its O(i, j) are recomputed, and only the ISG and SG offspring (i.e., except the VSG ones) are coded as pixels as given above.

In the refinement sub-pass, all the pixels in LL_M subband that are marked as VSG pixels are refined. Then, the LRS scanned for the SG roots only. For each SG root, its O(i, j) are also recomputed, and only the VSG ones are refined. A VSG pixel $c(i,j)$ is refined to a more bit precision by sending its bth bit to the bitstream. The coding pass terminates by updating T to $T/2$ to begin a new coding pass. To give users the option to recover the image at the desired quality and/or resolution, a scalable image compression system should encode the DWT image at the full bit rate and resolution. As a result, the encoder must keep going until all bits of all image pixels have been encoded (i.e., until $T = 1$). On the other hand, the decoder stops when the target bit-rate is attained.

III. The Proposed Algorithms

This section first introduces the modified SLS (MSLS). The MSLS algorithm has lower complexity, and slightly better performance than the SLS algorithm. Then, we present the proposed HS-MSLS, which is the main contribution of the paper. HS-MSLS adds resolution scalability to the MSLS to produce a highly scalable bitstream that is both quality and resolution scalable.

a) The MSLS Algorithm

The MSLS algorithm introduces the following two modifications to the SLS that lower its computational complexity and improve its performance especially at low bit-rates. The first modification is based on the observation that the maximum pixel value in the LL_M subband is 4-8 times the maximum pixel value in all the other subbands. This is because LL_M represents the average (DC) value of all the image pixels. So, the maximum bit-plane of LL_M ($b_{max}^{LL_M}$) is greater than the maximum bit-plane of all the other subbands (b_{max}^{Other}) by 2-3 or more bits. For instance, for the test images Lena, Goldhill, Mandrill, and Barbara, the value of $b_{max}^{LL_M}$ and b_{max}^{Other} are {12, 10}, {9, 6}, {12, 9}, and {12,10} respectively. Hence, as the threshold T depends on $b_{max}^{LL_M}$, then in the first 2 or 3 (or more) bit-plane coding passes, only the pixels in LL_M may be SG, and the pixels in all other subbands are ISG. Consequently, all the

SOTs are also ISG. Therefore, there will be a waste in the processing power, processing time, and the transmitted bits in attempting to test and code these ISG SOTs. The proposed solution is to employ two thresholds T_1 and T_2, such that $T_1 = 2^{b_{max}^{LL_M}}$, and $T_2 = 2^{b_{max}^{other}}$. The initial threshold T is set to T_1. Then, the algorithm performs several mini coding passes that encodes the pixels in $\mathrm{LL_M}$ subband only until $T = T_2$. At $T = T_2$, the algorithm performs the usual complete coding passes.

The second and the most important modification is the elimination of the need to recompute the four offspring that belong to every SG root during the sorting sub-pass and during the refinement sub-pass. The idea is based on that when a root $r(i,j)$ that is lied in levels R_0 to R_{M-2} (i.e., except R_{M-1}) becomes SG, its four offspring O(i, j) will be added to LRS. So, at the next coding passes, there is no need to recompute them again as they are already stored in LRS. That is, we need to recompute the offspring of the SG roots that are lied in the level R_{M-1} only instead of recomputing the offspring of the SG roots that are lied in levels R_0 to R_{M-1} as done in SLS. Evidently, this will reduce the algorithm's complexity. It worth noting that, according to this idea, every entry of LRS now plays the role of a root and a pixel.

Algorithm 1 gives the pseudo-code of the MSLS encoder, where W is the DWT image, $\mathrm{LL_M^X}$ represents the $\mathrm{LL_M}$ subband minus the excluded top-left pixel in each group of 2×2 pixels, and $\mathrm{LL_M^P}$ represents the excluded pixels in $\mathrm{LL_M}$. The MSLS is initialized by adding the (i, j) coordinates of the pixels in $\mathrm{LL_M^X}$ to LRS, and setting all state marker bits to 0. The encoder then performs the mini passes until $T = T_2$. In each one of these mini passes, every ISG or SG pixel in $\mathrm{LL_M}$ (i.e., with $\sigma(i,j) = 0\ or\ 1$) is encoded in the same manner done in SLS. The $Code_Pix(i,j)$ procedure given in algorithm 2 illustrates this step in details. Next, every pixel in $\mathrm{LL_M}$ that is marked as a VSP (i.e., with $\sigma(i,j) = 2$) in the $Code_Pix(i,j)$ during current or any one of the previous coding passes, is refined by the $Ref_Pix(i,j)$ procedure given in algorithm 3.

Starting from $T = T_2$, the algorithm implements the complete coding passes. Every pass begins by coding every pixel in $\mathrm{LL_M^P}$ as given above. Then, the LRS is encoded by three sub-passes. The first and the third sub-passes deal with the entries of LRS as pixels, while the second one deals with them as roots of the SOTs. In the first sub-pass, every ISG or SG entry (i,j) is encoded by the $Code_Pix(i,j)$. At the same time, if the corresponding root $r(i,j)$ is found SG in a previous pass, i.e., $\delta(i,j) = 1$, and it is lied in level R_{M-1}, then its O(i, j) are recomputed, and each ISG or SG one of them is encoded by the $Code_Pix(i,j)$. In the second sub-pass, every root $r(i,j)$ that is still ISG (i.e., with $\delta(i,j) = 0$), it is encoded in a manner similar to

that of the SLS algorithm. In the third pixel sub-pass, every (i,j) entry marked VSP during the current or a previous pass, is refined by the $Ref_Pix(i,j)$. At the same time if the root $r(i,j)$ is found SG in a previous pass, and lied in level R_{M-1}, then its O(i, j) are also recomputed, and each one of them marked as a VSP, is refined by the $Ref_Pix(i,j)$.

The MSLS algorithm is clarified using figure 2 which depicts a (16×16) pixels DWT image with M = 3 decomposition levels. The figure also shows the SOTs originated from the top-right pixel (marked by *), where every SOT is represented by a different color. LRS is initialized by the LL_3^X, which are three pixels (0,1), (1,0), and (1,1). In the mini passes, which are devoted to LL_3, the algorithm first sends the ISG and SG pixels in to the $Code_Pix(i,j)$ for encoding. Then, it sends the VSG pixels to the $Ref_Pix(i,j)$ for refining until $T = T_2$. At this instant, the encoder does the complete coding passes. Then, it sends pixels stored in LRS (which are lied in R_0) to the $Code_Pix(i,j)$. Next, it tests the roots in LRS to see if there are SG SOTs. Figure 3 depicts the content of the LRS during the 1st complete coding pass. Suppose that the root (0,1) has SG SOT (the yellow one). So, O(0,1) = {(0,2), (0,3), (1,2), (1,3)} are sent to the $Code_Pix(c_{i,j})$, and added to LRS because the parent root (0,1), is lied in R_0 (LL_3). The added offspring are considered as roots to processed exactly as their parent roots. Suppose again that the roots (0,2), and (1,3) have SG SOTs. So, O(0,2) = {(0,2), (0,3), (1,2), (1,3)}, and O(1,3) = {(2,6), (2, 7), (3,6), (3,7)}, are also sent to the $Code_Pix(i,j)$, and added to LRS because roots (1,2), and (1,3) are lied in R_1 (HL_3). Suppose finally that the root (1,4) has a SG SOT, so O(1,4) = {(2, 8), (2, 9), (3,8), (3,9)} are sent to the $Code_Pix(i,j)$, but they will not be added to LRS because the root (1,4) is lied in R_2 (i.e., its offspring are the leaves of the SOT that are lied in R_3). During the refinement sub-pass, only the pixels (0,1), (1,0), and (1,1) may be VSG pixels to be passed to $Ref_Pix(i,j)$ because they are previously encoded during the mini passes. All other pixels are either ISG or just become SG. It is worth noting that during the next coding pass, all the current pixels in LRS are sent to the $Code_Pix(i,j)$. In addition, the root (1,4) is lied in R_2, and found SG at the 1st pass. So, its four offspring are recomputed because they were not added to LRS in the last pass. Notice that, for this case, the size of LRS is equal to (16×16)/4 = 64 entry.

In comparison to SLS, it was necessary to recompute the offspring of every SG root in LRS two times in every coding pass. For this example, there are four SG roots with total offspring recomputing equal to 8 times. On the other hand, MSLS needs to recompute the offspring two times of the SG roots that are located in R_{M-1} only

which is one root with total offspring recomputing equal to 2 times. The reduction in the algorithm's complexity is obvious.

1. Initialization
   - $2^{b_{max}^{LL_M}} = \left\lfloor log_2 \{\max_{\forall(i,j)\in LL_M} |c(i,j)|\} \right\rfloor$;
   - $2^{b_{max}^{other}} = \left\lfloor log_2 \{\max_{\forall(i,j)\in W-LL_M} |c(i,j)|\} \right\rfloor$;
   - $send\ 2^{b_{max}^{LL_M}}\ and\ 2^{b_{max}^{other}}\ to\ bitstream$;
   - $T_1 = 2^{b_{max}^{LL_M}};\ T_2 = 2^{b_{max}^{other}};T = T_1$;
   - $add\ all\ (i,j) \in LL_M^X\ to\ LRS$;
2. Coding the mini passes
   $while\ T \geq T_2\ do$:
   2.1 $\forall\ (i,j) \in LL_M\ do$:
       $if\ \sigma(i,j) = 0\ or\ 1\ do: Code_Pix(i,j)$;
   2.2 $\forall\ (i,j) \in LL_M\ do$:
       $if\ \sigma(i,j) = 2\ do: Ref_Pix(i,j)$;
   2.3 $T = T/2$;
3. The Coding passes
   3.1 $T = T_2$;
   3.2 $\forall\ (i,j) \in LL_M^P\ do: do\ step$ 2.1 & 2.2;
   3.3 $\forall\ (i,j) \in LRS\ do$:
   - $if\ \sigma(i,j) = 0\ or\ 1\ do$:
     $Code_Pix(i,j)$;
   - $if\ \delta(i,j) = 1\ \&\ (i,j) \in R_{M-1}\ do$:
     - $compute\ O(i,j)$;
     - $\forall\ O(i,j) \in (i,j)\ do$:
       - $if\ \sigma(O(i,j)) = 0\ or\ 1\ do: Code_Pix(i,j)$;

   3.4 $\forall\ (i,j) \in LRS\ do$:
   - $if\ \sigma(i,j) = 2\ do: Ref_Pix(i,j)$;
   - $if\ \delta(i,j) = 1\ \&\ (i,j) \in R_{M-1}\ do$:
     - $compute\ O(i,j)$;
     - $\forall\ O(i,j) \in (i,j)\ do$:
       - $if\ \sigma(O(i,j)) = 2\ do: Ref_Pix(i,j)$;

   3.5 $\forall\ (i,j) \in LRS\ do$:
   - $if\ \delta(i,j) = 0\ do$:
     - $if\ SOT(i,j)\ is\ SG\ do$:
       - $send\ 1\ to\ bitstream$;
       - $\delta(i,j) = 1$;
       - $add\ O(i,j)\ to\ LRS$;
       - $\forall\ O(i,j)\ do: Code_Pixel(O(i,j))$;
     - $else\ do: send\ 0\ to\ bitstream$;

   3.6 $T = T/2; and\ goto\ step\ 3.2\ if\ needed$;

Algorithm 1: The MSLS encoder

$Code_Pix(i,j)\{$

- $if\ \sigma(i,j) = 0\ do$:
  - $if\ |c(i,j)| \geq T\ do$:
    - $send\ 1\ \&\ sign\ bit\ of\ c(i,j)\ to\ bitstream$;
    - $\sigma(i,j) = 1$;
    - $if\ c(i,j) > 0\ do$:
      $c(i,j) = c(i,j) - T$;
    - $elseif\ c(i,j) < 0\ do$:
      $c(i,j) = c(i,j) + T$;
  - $elseif\ |c(i,j)| < T\ do$:
    $send\ 0\ to\ bitstream$;
- $elseif\ \sigma(i,j) = 1\ do: \sigma(i,j) = 2;\}$

Algorithm 2: The $Code_Pix(i,j)$ procedure

$Ref_Pix(i,j)\{$

- $if\ |c(i,j)| \geq T\ do$:
  - $send\ 1\ to\ bitstream$;
  - $if\ c(i,j) > 0\ do$:
    $c(i,j) = c(i,j) - T$;
  - $elseif\ c(i,j) < 0\ do$:
    $c(i,j) = c(i,j) + T$;
- $elseif\ |c(i,j)| < T\ do$:
  $send\ 0\ to\ bitstream;\}$

Algorithm 3: The $Ref_Pix(i,j)$ procedure

Columns 0–15 / Rows 0–15

×
*
LL_3
HL_3
LL_3
HH_3
HL_2
HL_1
LH_2
HH_2
LH_1
HH_1

Figure 2. Part of the SOTs for a (16×16) pixels 2D-DWT image with three decomposition levels

Resolution level	LRS index	LRS content during the 1st complete coding pass	
R_0	0	(0,1)	
	1	(1, 0)	
	2	(1,1)	
R_1	3	(0,2)	$O(0,1)$
	4	(0, 3)	
	5	(1,2)	
	6	(1,3)	
R_2	7	(0,4)	$O(0,2)$
	8	(0,5)	
	9	(1,4)	
	10	(1,5)	
	11	(2,6)	$O(1,3)$
	12	(2,7)	
	13	(3,6)	
	14	(3,7)	
	19		
	⋮		
	63		

(a)

Resolution level	LRS index	LRS content during the 2nd complete coding pass	
R_0	0	(0,1)	
	1	(1, 0)	
	2	(1,1)	
R_1	3	(0,2)	
	4	(0, 3)	
	5	(1,2)	
	6	(1,3)	
R_2	7	(0,4)	
	8	(0,5)	
	9	(1,4)	
	10	(1,5)	
	11	(2,6)	
	12	(2,7)	
	13	(3,6)	
	14	(3,7)	
R_1	15	(2,0)	$O(1,0)$
	16	(2,1)	
	17	(3,0)	
	18	(3,1)	
R_2	19	(2,4)	$O(1,2)$
	20	(2,5)	
	21	(3,4)	
	22	(3,5)	
	23		
	⋮		
	63		

(b)

Figure 3: The content of the LRS. a) at the 1st complete pass; b) at the 2nd complete pass

1. Initialization
- $Receive\ 2^{b_{max}^{LL_M}}\ and\ 2^{b_{max}^{other}}\ from\ bitstream;$
- $T_1 = 2^{b_{max}^{LL_M}};\ T_2 = 2^{b_{max}^{other}};T = T_1;$
- $add\ all\ (i,j) \in LL_M^X\ to\ LRS;$

2. Decoding the mini passes
   $while\ T \geq T_2\ do:$
   2.1 $\forall\ (i,j) \in LL_M\ do:$
   $if\ \sigma(i,j) = 0\ or\ 1\ do: Decode_Pix(i,j);$
   2.2 $\forall\ (i,j) \in LL_M\ do:$
   $if\ \sigma(i,j) = 2\ do: Deref_Pix(i,j);$
   2.3 $T = T/2;$
3. The Coding passes
   3.1 $T = T_2;$
   3.2 $\forall\ (i,j) \in LL_M^P\ do: do\ step\ 2.1\ \&\ 2.2;$
   3.3 $\forall\ (i,j) \in LRS\ do:$
   - $if\ \sigma(i,j) = 0\ or\ 1\ do: Decode_Pix(i,j);$
   - $if\ \delta(i,j) = 1\ \&\ (i,j) \in R_{M-1}\ do:$
     - $compute\ O(i,j);$
     - $\forall\ O(i,j) \in (i,j)\ do:$
       - $if\ \sigma(O(i,j)) = 0\ or\ 1\ do: Decode_Pix(i,j);$

   3.4 $\forall\ (i,j) \in LRS\ do:$
   - $if\ \sigma(i,j) = 2\ do: Deref_Pix(i,j);$
   - $if\ \delta(i,j) = 1\ \&\ (i,j) \in R_{M-1}\ do:$
     - $compute\ O(i,j);$
     - $\forall\ O(i,j)\ \in (i,j)\ do:$
       - $if\ \sigma(O(i,j)) = 2\ do: Deref_Pix(i,j);$

   3.5 $\forall\ (i,j) \in LRS\ do:$
   - $if\ \delta(i,j) = 0\ do:$
     - $if\ bitstream = 1\ do:$
       - $\delta(i,j) = 1;$
       - $add\ O(i,j)\ to\ LRS;$
       - $\forall\ O(i,j)\ do: Decode_Pixel(O(i,j));$

   3.6 $T = T/2; and\ goto\ step\ 3.2\ if\ needed;$

Algorithm 4: The MSLS decoder

$Decode_Pix(i,j)\{$
- $if\ \sigma(i,j) = 0\ do:$
  - $if\ bitstream = 1$
    - $\sigma(i,j) = 1;$
    - $receive\ sign\ bit\ from\ bitstream;$
    - $if sign\ bit = 0\ do: c(i,j) = 1.5T;$
    - $else\ do: c(i,j) = -1.5T;$
- $elseif\ \sigma(i,j) = 1\ do: \sigma(i,j) = 2;\ \}$

Algorithm 5: The $Decode_Pix(i,j)$ procedure

$Deref_Pix(i,j)\{$
- $if\ bitstream = 1$
  - $if c(i,j) > 0\ do:$
    $c(i,j) = c(i,j) + \lfloor T/2 \rfloor;$
  - $elseif\ c(i,j) < 0\ do:$
    $c(i,j) = c(i,j) - \lfloor T/2 \rfloor;$
- $else\ do:$
  - $if c(i,j) > 0\ do:$
    $c(i,j) = c(i,j) - \lfloor T/2 \rfloor;$
  - $elseif\ c(i,j) < 0\ do:$
    $c(i,j) = c(i,j) + \lfloor T/2 \rfloor;\ \}$

Algorithm 6: The $Deref_Pix(i,j)$ procedure

The decompression algorithm undoes the same steps of the encoder. However, it doesn't need to do the significance test for the pixels and the SOTs. That is, when the decoder receives 0/1, this means that the corresponding pixel or SOT is ISG/SG respectively. So, the decoder doesn't need to build the SOTs and check their significance as the corresponding received bit determines this. This leads to make the decompression algorithm runs faster than the compression one. This feature is very valuable for scalable image compression schemes since images are compressed once but may be decompressed several times [3, 21]. Algorithms 4, 5, and 6 gives

the pseudo-codes of the MSLS decoder, the $Decode_Pix(i,j)$ procedure, and the $Deref_Pix(i,j)$ procedure respectively. A part from the above note, the steps of the MSLS encoder and decoder are identical. However, the $Decode_Pix(i,j)$ and the $Deref_Pix(i,j)$ differ from the $Encode_Pix(i,j)$ and $ref_Pix(i,j)$ by the pixel reconstruction process. At the decoder, the initial value of all image pixels is 0. During the $Decode_Pix(i,j)$, if $c(i,j)$ is yet ISG ($\sigma(i,j) = 0$), and the received bit = 1, then $c(i,j)$ becomes SG. So $\sigma(i,j)$ is set to 1, and $c(i,j)$ is updated to ±1.5×T depending on the received sign bit (0 for positive, and 1 for negative). On the other hand, if $c(i,j)$ is found SG in one of the previous coding passes ($\sigma(i,j) = 1$), it is updated to VSG ($\sigma(i,j) = 2$) to be refined. During the $Deref_Pix(i,j)$, $c(i,j)$ is refined by updating its value to $c(i,j) \mp \lfloor T/2 \rfloor$ depending on the received bit and its sign as given in the procedure. Table 1 describes the encoding and decoding processes for $c(i,j) = 45$ and with initial $T = 32$. In the first coding pass, the pixel $c(i,j)$ is processed by $Encode_{Pix(i,j)}/Decode_Pix(i,j)$, while in the other coding passes, it is processed by $ref_{Pix(i,j)}/Deref_Pix(i,j)$ respectively.

Table 1: The encoding and decoding processes for $c(i,j) = 45$ and $T = 32$

Coding pass	T	$c(i,j)$	New $c(i,j)$	Reason	Bitstream	Recovered $\hat{c}(i,j)$
1	32	45	45-32 = 13	45 ≥ 32	1, 0	1.5×32 = 48
2	16	13	13	13 < 16	0	$48 - \lfloor 16/2 \rfloor = 40$
3	8	13	13-8 = 5	13 ≥ 8	1	$40 + \lfloor 8/2 \rfloor = 44$
4	4	5	5-4 = 1	5 ≥ 4	1	$44 + \lfloor 4/2 \rfloor = 46$
5	2	1	1	1 < 2	0	$46 - \lfloor 2/2 \rfloor = 45$
6	1	1	1-1 = 0	1 ≥ 1	1	$45 + \lfloor 1/2 \rfloor = 45$

b) The HS-MSLS Algorithm

As stated before, a highly scalable bitstream must be both quality and resolution scalable. So, a QSIC algorithm like MSLS can be upgraded to be HSIC by encoding the resolution levels ($R_0 - R_M$) in each coding pass successively. That is, each coding pass must encode all the data that belongs to R_m before proceeding to the next level R_{m+1}. In addition, sufficient resolution tag markers G_m, $0 \le m \le M$, must be added to the bitstream to identify the different resolution levels. G_m represents the length (in bytes) of the bitstream devoted to the resolution level R_m.

It can be easily shown that the coding paradigm of the MSLS algorithm permits to attain this object partially. Let K_m be the number of pixels stored in LRS that belongs to the resolution level R_m, $0 \le m \le M-1$. Clearly K_0 is fixed and equal to the number of pixels in the LL_M^X subband which is equal to $K_0 = \frac{3}{4}|LL_M| =$

$\frac{3}{4}\left|\mathrm{N} \times \mathrm{N}/2^{2\mathrm{M}}\right|$ for an (N×N) pixels DWT image with M decomposition levels. For our example, $K_0 = \frac{3}{4}|16 \times 16/2^6| = 3$ pixels. So, we can see that each coding pass already encodes the data that belong to R_0 level first. Referring back to figure 3a, we can see that during the first coding pass, the data of LRS is also stored in increasing order of resolution levels. However, K_m, of each level is not fixed as it depends on the number of the roots that have SG SOTs. This problem can be solved simply by counting and storing K_m, m = 1, 2 …M−1. For our example, $K_1 = 4$, and $K_2 = 8$.

The problem becomes more complicated apart from the second pass. Continuing with the same example, and assume that at this pass, the roots (1,0), and (1,2) have SG SOTs (the green ones shown in Figure 3b). So, O(1,0) = {(2,0), (2,1), (3,0), (3,1)} that are lied in R_1, and O(1,2) = {(2,4), (2,5), (3,4), (3,5)}, that are lied in R_2, are added to the end of LRS. As shown, LRS is not more arranged according to the resolution levels. The previous solution is not sufficient as we must know the (separated) portions where the data of the different levels are located in addition to their sizes. For instance, to encode the data of R_1, we must encode the portion of data at indices (3-6), and the portion of data at indices (15-18). The same must be done for R_2. It is worth noting that during each one of the next passes, the number of separated portions for every resolution level increases. Thus, it will be difficult to track them.

The proposed solution to this problem is to preserve contiguous fixed-size portions within the LRS for the data of the different resolution levels referred to as resolution-dependent portions P_m such that P_m is the portion in LRS devoted to store the data that belong to resolution level $R_m, 0 \leq m \leq M-1$. In order to guarantee that P_m can support all the pixels of R_m, its size must equal to the maximum size of R_m (k_m^{max}). Since K_0 is fixed, so $k_0^{max} = K_0$. It can be shown that for $m \geq 1, k_m^{max}$ is equal to the total number of pixels in the three subbands $\mathrm{HL}_{\mathrm{M-m+1}}$, $\mathrm{LH}_{\mathrm{M-m+1}}$, and $\mathrm{HH}_{\mathrm{M-m+1}}$ that constitute R_m. As these subbands are of equal sizes, so, $k_m^{max} = 3|\mathrm{HL}_{\mathrm{M-m+1}}|$. To track these portions, we use two pointers referred to as the portion start pointer PS_m and the portion end pointer PE_m. PS_m stores the index of the first pixel, and PE_m stores the index of the last pixel in P_m respectively. Evidently, $PS_0 = 0$, and $PE_0 = k_0^{max} - 1$. For $m \geq 1$, $PS_m = PS_{m-1} + k_{m-1}^{max}$. PE_m is initialized by PS_m, and it is updated each time a pixel is added to P_m. Referring back to our case, $k_0^{max} = 3$ pixels, so $PS_0 = 0$, and $PE_0 = 3 - 1 = 2$. Similarly, $PS_1 = PE_1 = PS_0 + k_0^{max} = 0 + 3 = 3$. Lastly, $k_1^{max} = 3|LH_3| = 12$ pixels, so $PS_2 = PE_2 = PS_1 + k_1^{max} = 3 + 12 = 15$.

Resolution level & portion	LRS index	LRS content during the 1st coding pass	
R_0 P_0	0	(0,1)	
	1	(1, 0)	
	2	(1,1)	
R_1 P_1	3	(0,2)	$O(0,1)$
	4	(0, 3)	
	5	(1,2)	
	6	(1,3)	
	7		
	8		
	9		
	10		
	⋮		
	14		
R_2 P_3	15	(0,4)	$O(0,2)$
	16	(0,5)	
	17	(1,4)	
	18	(1,5)	
	19	(2,6)	$O(1,3)$
	20	(2,7)	
	21	(3,6)	
	22	(3,7)	
	23		
	24		
	⋮		
	255		

(a)

Resolution level & portion	LRS index	LRS content during the 2nd coding pass	
R_0 P_0	0	(0,1)	
	1	(1, 0)	
	2	(1,1)	
R_1 P_1	3	(0,2)	$O(0,1)$
	4	(0, 3)	
	5	(1,2)	
	6	(1,3)	
	7	(2,0)	$O(1,0)$
	8	(2,1)	
	9	(3,0)	
	10	(3,1)	
	⋮		
	14		
R_2 P_3	15	(0,4)	$O(0,2)$
	16	(0,5)	
	17	(1,4)	
	18	(1,5)	
	19	(2,6)	$O(1,3)$
	20	(2,7)	
	21	(3,6)	
	22	(3,7)	
	23	(2,4)	$O(1,2)$
	24	(2,5)	
	25	(3,4)	
	26	(3,5)	
	⋮		
	255		

(b)

Figure 4: The content of the adopted resolution-dependent portions LRS used by the proposed HS-MSLS algorithm. a) at the 1st complete pass; b) at the 2nd complete pass.

The adopted structure of LRS facilitates sorting and tracking the roots and their offspring according to the resolution level they belong. This is simply achieved: if a root $r(i,j)$ stored at the portion $P_m, 0 \leq m \leq M-2$, has a SG SOT, then its four offspring O(i, j) are added at the end of the next portion P_{m+1}. Notice that for the roots that are lied at the portion P_{M-1}, their offspring are not added to LRS. These offspring will be deduced from their parent roots that are lied at the portion P_{M-2}.

Figures 4a, and 4b depict the structure of the adopted resolution-dependent portions LRS for the 1st and 2nd complete coding passes respectively for the same example. As shown in figure 4a, the root (0,1) is lied in P_0, so O(0,1) = {(0,2), (0,3), (1,2), (1,3)}are added to P_1 starting from index $PE_1 = 3$, and PE_1 is updated to 3+4 = 7. The roots (0,2), and (1,3) are lied in P_1, so O(0,2) = {(0,4), (0,5), (1,4), (1,5)}, and

$O(1,3)$ = {(2,6), (2,7), (3,6), (3,7)} are added to P_2 starting from index $PE_2 = 15$, and PE_2 is updated to 15+4+4 = 23. At the 2nd pass, the root (1,0) is lied in P_0, so $O(1,0)$ = {(2,0), (2,1), (3,0), (3,1)}are added to P_1 starting from index $PE_1 = 7$, and PE_1 is updated to 7+4 = 11. Finally, the root (1,2) is lied P_1, so $O(1,2)$ = {(1,4), (1,5), (2,4), (2,5)} are added to P_2 starting from index $PE_2 = 23$, and PE_2 is updated to 27.

The HS-MSLS algorithm works exactly as MSLS except that it makes use of resolution-dependent coding passes with the associated LRS resolution-dependent portions. The mini passes, which are devoted to LL_M (R_0), starts by putting the resolution tag G_0 in the bitstream, followed by passing the ISG and SG pixels to the $Code_Pix(i,j)$, and the VSG pixels to the $Ref_Pix(i,j)$. Every complete coding pass also starts by putting G_0 followed by passing the ISG and SG pixels in LL_M^P, and the ISG and SG entries in P_0 of LRS to the $Code_Pix(i,j)$. Then, it performs the pixel sorting sub-pass of LRS for each portion P_m, $1 \leq m \leq M-1$. This is achieved by putting G_m in the bitstream, followed by passing the ISG and SG entries in P_m to the $Code_Pix(i,j)$. Next, G_M is put in the bitstream, and for every root $r(i,j)$ found SG in a previous pass and belongs to P_{M-1}, its O(i, j) are recomputed and ISG and SG ones are passed to the $Code_Pix(i,j)$. The root pass of LRS is also performed for each portion P_m by putting G_{m+1} in the bitstream. Then, every ISG root is processed exactly as done in MSLS. Notice that the resolution tag is started at G_{m+1} because this pass encodes the root's offspring, which are lied in R_{m+1} for a root that is lied in R_m. Lastly, the refinement pixel sub-pass of LRS is also done for each resolution portion $P_m, 0 \leq m \leq M-1$ by putting G_m in the bitstream, followed by sending the VSG entries in P_m to the $Ref_Pix(i,j)$. Finally, G_M is put in the bitstream, and for every root $r(i,j)$ found SG in a previous pass and belongs to P_{M-1}, its O(i, j) are recomputed and each VSG one of them is sent to the $Ref_Pix(i,j)$. As mentioned before, G_m represents the length of the coded data of R_m. Obviously, G_m will be available only after finishing coding R_m. So, at least the bitstream of the corresponding resolution level must be buffered until its length is available.

Figure 5 depicts the structure of the bitstream of the HS-MSLS algorithm for the first coding pass. The other coding passes have the same structure as the first one excluding the part devoted to the mini-passes. The header contains information about the image such as image name, image size, $b_{max}^{LL_M}$, b_{max}^{other}, etc. The bitstream is divided into four parts. The mini-passes part consists of the resolution level part R_0 only. The other three parts consist of M resolution levels parts. Each R_m consists of the resolution tag G_m, and the output bitstream due to coding the portion P_m within LRS started from SE_m to PE_m.

During the decoding, an image at resolution R_m, $0 \leq m \leq M$ can be reconstructed by parsing the bitstream, and in each bit-plane coding pass the data that belongs to $R_0 - R_m$ are selected and the rest of the data are skipped. Then, an m stages of inverse 2D-DWT is done only. The size of the reconstructed image is equal to $1/2^{2m}$ the size of the original image.

	R_0		R_0			R_1		R_2		…	R_{M-1}		R_M	
Header	G_0	LL_M	G_0	LL_M^P	P_0	G_1	P_1	G_2	P_2	…	G_{M-1}	P_{M-1}	G_M	P_{M-1}
	The Mini-passes		The first LRS sub-pass (sorting sub-pass for pixels)											
R_1			R_2		…	R_M		R_0		R_1		…	R_M	
G_1	P_0		G_2	P_1	…	G_M	P_{M-1}	G_0	P_0	G_1	P_1	…	G_M	P_M
The second LRS sub-pass (sorting sub-pass for roots)								The third LRS sub-pass (refinement sub-pass)						

Figure 5: The bitstream structure of the HS-MSLS for the mini passes and the first complete pass

IV. Simulation Results and Discussion

The proposed MSLS and the HS-MSLS algorithms are evaluated by MATLAB Package using a laptop furnished by Intel Core i3 μprocessor with 1.8 GHz CPU and 2 GB RAM. We employed the conventional gray-scale test images Lena, Barbara, Mandrill, and Goldhill, each of size (512×512) pixels. The images are first transformed using the dyadic (9, 7) 2D-DWT. The results are represented by the algorithm's performance, its computational complexity, and its memory usage against the compression bit-rate which is the average number of bits per pixel (bpp) for the compressed image.

The performance is measured by the mean squared error (MSE) between the original image (I_o), and the reconstructed image (I_r), each of size M×N pixels. MSE is defined as:

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

However, the Peak Signal to Noise Ratio (PSNR), which is derived from the MSE, is more employed. It is defined as:

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

where P_{max} is the maximum pixel value in I_o. For grayscale images, $P_{max} = 255$. Obviously, the lowest MSE, or the highest PSNR for a given bpp is the target.

a) PSNR Performance of the QSIC Algorithms

Table 2 gives the PSNR against the bit-rate for the QSIC SPIHT [7], SLS [17], and the proposed MSLS algorithms. The results of SPIHT are gotten by executing the SPIHT Public License MATLAB program of Mustafa, and Pearlman [22]. Every image is decomposed into M = 6 levels since SPIHT is evaluated using this value of M. The value of PSNR where MSLS is the highest is made boldface. As clearly shown, the PSNR superiority of MSLS over SPIHT is apparent for all the test images, and at all bit-rates. Moreover, MSLS is slightly better than SLS. This improvement is achieved mainly due to the adopted two thresholds method that reduced the number of transmitted bits especially at the early bit-planes coding passes.

Table 2: PSNR versus bit-rate of the QSIC algorithms

Bit rate (bpp)	PSNR (dB)					
	Lena			Goldhill		
	SPIHT	SLS	MSLS	SPIHT	SLS	MSLS
0.0625	26.51	27.56	**27.59**	26.17	26.17	**26.25**
0.125	29.39	30.15	**30.18**	27.77	27.76	**27.84**
0.25	32.71	33.05	**33.07**	29.75	29.76	**29.79**
0.5	36.13	36.30	**36.31**	32.27	32.29	**32.31**
1	39.50	39.61	**39.62**	35.51	35.57	**35.58**
Bit rate (bpp)	PSNR (dB)					
	Barbara			Mandrill		
	SPIHT	SLS	MSLS	SPIHT	SLS	MSLS
0.0625	23.07	23.44	**23.46**	19.00	20.35	**20.35**
0.125	24.36	24.69	**24.71**	19.54	21.32	**21.33**
0.25	26.96	27.37	**27.39**	20.55	22.69	**22.70**
0.5	30.97	31.14	**31.17**	23.14	24.88	**24.89**
1	36.25	36.37	**36.37**	28.41	28.33	**28.33**

b) PSNR Performance of the HSIC Algorithms

Table 3 depicts the PSNR against the bit-rate for the HSIC LHS-SPIHT [18], HSLS [15], and the proposed HS-MSLS algorithms when the decoder recovers the image at full resolution (m = 5). We used M = 5 decomposition levels as the other algorithms. The result of MSLS is also included in order to investigate the effect of

making the algorithm highly scalable on its PSNR. As the table indicate, the proposed HS-MSLS algorithm has the highest PSNR for nearly all cases. In addition, HS-MSLS has unnoticeable PSNR deterioration as compared to MSLS. This is very normal due to adding the resolution tag indicators to the bitstream, and due to coding the resolution levels in increasing order.

Table 3: PSNR versus bit-rate of the HSIC algorithms at full resolution (M = 5, m = 5)

Bit rate (bpp)	PSNR (dB)							
	Lena				Goldhill			
	LHS-SPIHT	HSLS	HS-MSLS	MSLS	LHS-SPIHT	HSLS	HS-MSLS	MSLS
0.0625	26.85	27.35	**27.43**	27.52	26.26	26.15	26.22	26.25
0.125	29.93	30.04	**30.08**	30.12	27.50	27.80	**27.82**	27.84
0.25	33.19	33.00	**33.00**	33.04	29.39	29.73	**29.75**	29.79
0.5	36.49	36.24	**36.25**	36.30	32.10	32.05	**32.23**	32.31
1	39.58	39.58	**39.59**	39.61	35.54	35.40	35.53	35.58
Bit rate (bpp)	PSNR (dB)							
	Barbara				Mandrill			
	LHS-SPIHT	HSLS	HS-MSLS	MSLS	LHS-SPIHT	HSLS	HS-MSLS	MSLS
0.0625	22.59	23.37	23.41	**23.46**	20.38	20.26	20.27	20.35
0.125	23.65	24.26	24.28	**24.71**	21.25	21.27	**21.30**	21.33
0.25	26.75	27.31	27.33	**27.39**	22.66	22.58	**22.67**	22.70
0.5	30.48	31.05	31.07	**31.17**	24.60	24.68	**24.71**	24.89
1	35.19	36.23	36.28	**36.37**	28.30	28.30	**28.30**	28.33

As mentioned previously, with a HSIC, the decoder may reconstruct the image at different resolution than that of the original image. So, we can't use equations 2 and 3 directly to compute the PSNR because the original and recovered images have different sizes. Danyali et. al [19] solved this problem by exploiting the fact that an image recovered at resolution m, $0 \leq m \leq M$, represents the LL_{M-m} subband of the 2D-DWT image. So, the original LL^{o}_{M-m} and the recovered LL^{r}_{M-m} subbands, which have the same size, are used instead of the full-size original and recovered images. For our case, an image size (512 × 512) pixels is decomposed with M = 5 levels, and if the image is recovered at resolution m = 4, then $LL^{o}_{M-m} = LL^{o}_{5-4} = LL^{o}_{1}$ represents the original image, and LL^{r}_{1} represents the recovered image, each has (256 × 256) pixels (1/4 the original image size).

Tables 4 and 5 shows the PSNR vs. the bit-rate when the image recovered at 1/4 resolution (M = 5, m = 4), and at 1/16 resolution (M = 5, m = 3) respectively. The bit-rate is calculated with respect to the size of the original image. The PSNR improvement of the HS-MSLS over the other algorithms is very clear. There is one exception, in table 5, at 0.5 bpp. The PSNR of HS-MSLS is lower than that of LHS-

SPIHT for all images except Mandrill. It should be noted that the number with the PSNR cell after the (@) sign represents the full bit-rate. That is, all the bits of all image pixels are encoded. As shown, we get lower PSNR but at lower bit-rate. However, a PSNR ≥ 50 dB can be considered of a perfect quality [23].

Table 4: PSNR versus bit-rate of the HSIC algorithms at 1/4 resolution (M = 5, m = 4)

Bit rate (bpp)	PSNR (dB)					
	Lena			Goldhill		
	LHS-SPIHT	HSLS	HS-MSLS	LHS-SPIHT	HSLS	HS-MSLS
0.0625	27.98	28.45	**28.57**	27.36	27.61	**27.72**
0.125	31.84	32.14	**32.21**	29.78	30.21	**30.25**
0.25	37.21	37.01	37.15	32.97	32.79	32.79
0.5	43.52	43.35	43.38	38.51	38.62	**38.68**
1	53.17	53.05	**53.34**	49.73	49.77	49.67
Bit rate (bpp)	PSNR (dB)					
	Barbara			Mandrill		
	LHS-SPIHT	HSLS	HS-MSLS	LHS-SPIHT	HSLS	HS-MSLS
0.0625	25.48	26.84	**26.92**	21.28	22.74	**22.85**
0.125	27.93	29.24	**29.35**	23.45	24.73	**24.42**
0.25	32.42	33.66	**33.72**	28.59	28.87	**27.59**
0.5	38.97	39.23	**40.53**	31.21	31.58	**32.99**
1	50.02	50.19	**51.07**	39.23	39.52	**43.58**

Table 5: PSNR versus bit-rate of the HSIC algorithms at 1/16 resolution (M = 5, m = 3)

Bit rate (bpp)	PSNR (dB)					
	Lena			Goldhill		
	LHS-SPIHT	HSLS	HS-MSLS	LHS-SPIHT	HSLS	HS-MSLS
0.0625	31.86	32.08	**32.44**	30.86	31.33	**31.50**
0.125	39.53	40.34	39.92	36.12	36.87	**37**
0.25	50.30	50.89	**50.98**	46.87	47.05	**47.98**
0.5	70.51	64.77 @0.45	64.77 @0.45	70.71	64.80 @0.47	64.80 @0.47
Bit rate (bpp)	PSNR (dB)					
	Barbara			Mandrill		
	LHS-SPIHT	HSLS	HS-MSLS	LHS-SPIHT	HSLS	HS-MSLS
0.0625	29.89	31.93	**32.31**	25.21	26.00	**28.20**
0.125	35.83	36.03	**37.80**	31.31	30.42	**32.88**
0.25	46.12	46.52	**48.53**	40.82	40.87	**43.68**
0.5	70.89	63.75 @0.46	64.84 @0.47	59.97	64.70	64.77

Figure 7 shows the image Lena reconstructed at 0.25 bpp, at full, 1/4, and 1/16 resolutions with the corresponding PSNR 33, 37.15, and 50.98 dB, respectively. As depicted, for the same bit-rate we can get better PSNR by reconstructing the image at a reduced resolution. Inversely, we can preserve the same PSNR while the bit-rate is reduced if the image is reconstructed at reduced resolution.

(a) PSNR = 33 dB (b) PSNR = 37.15 dB (c) PSNR = 50.98 dB

Figure 7: Lena image reconstructed at 0.25 bpp. a) at full resolution; b) at 1/4 resolution; and c) at 1/16 resolution

b) Computational Complexity

Tables 6 and 7 show the complexity represented by the encoding time and the decoding time against the bit-rate for the different QSIC, and HSIC algorithms respectively. The MSLS is also included in table 7 to investigate the effect of making the algorithm highly scalable on its complexity. The Lena image is selected for this purpose. The shortest coding and decoding times at each bit-rate is made boldface. As it can be noticed, for all algorithms, the encoding time is longer than the decoding time. This is expected since the decoder does not require building the SOTs and testing their significances. From table 6, it can be shown that for MSLS, both the encoding and decoding times are widely shorter than that of SPIHT for all bit-rates. This is mainly due to removing the lists, which in turn reduced the random access read/write memory operations. Additionally, these times are also shorter than that of SLS. This speed improvement is the result of eliminating the need of offspring recalculation for the SG roots that are lied in resolution levels R_0–R_{M-2} twice in each coding pass.

Table 7 shows that our HS-MSLS algorithm is again the fastest one in encoding and decoding times among the other HSIC algorithms. In addition, a comparison between MSLS and HS-MSLS reveals that the later encompasses unnoticeable speed increment in encoding and decoding. The reason for this negligible increment is separating the LRS into the M resolution portions (P_m) that are FIFO accessed by the associated M pointers instead of FIFO accessing the entire LRS by one pointer.

Table 6: The encoding time and the decoding time
of the QSIC algorithms versus the bit-rate for Lena image

Bit rate	Encoding time (seconds)			Decoding time (seconds)		
(bpp)	SPIHT	SLS	MSLS	SPIHT	SLS	MSLS
0.0625	0.750	0.734	**0.359**	0.391	0.078	**0.031**
0.125	1.281	0.765	**0.531**	0.672	0.093	**0.062**
0.25	2.078	1.062	**0.875**	1.000	0.188	**0.140**
0.5	3.328	1.609	**1.359**	1.828	0.223	**0.186**
1	5.719	2.421	**2.156**	3.500	0.556	**0.348**

Table 7: The encoding time and the decoding time
versus the bit-rate of the HSIC algorithms for Lena image

Bit rate (bpp)	Encoding time (seconds)				Decoding time (seconds)			
	LHS-SPIHT	HSLS	HS-MSLS	MSLS	LHS-SPIHT	HSLS	HS-MSLS	MSLS
0.0625	0.579	0.495	**0.360**	0.359	0.141	0.105	**0.032**	0.031
0.125	0.814	0.611	**0.533**	0.531	0.157	0.146	**0.064**	0.062
0.25	1.095	1.062	**0.877**	0.875	0.203	0.189	**0.142**	0.140
0.5	1.593	1.609	**1.363**	1.359	0.234	0.204	**0.189**	0.186
1	2.359	2.578	**2.159**	2.156	0.382	0.362	**0.351**	0.348

c) Memory requirements

The memory requirement is measured by the amount of computer memory needed by the algorithm to compress/decompress an image with (N×N) pixels. As mentioned previously, the memory of SPIHT is variable as it depends on the bit-rate. However, in order to guarantee that SPIHT works properly for all bit-rates, the memory that is required to compress/decompress at full bit-rate must be used. At this rate total number of LSP and LIP entries is equal to two times the number of pixels (N×N), and the number of LIS entries is equal to N×N/4 [10]. So, the total memory of SPIHT is:

$$MEM_{SPIHT} = 2b(2N^2) + 2c\left(\frac{N^2}{4}\right) = 4bN^2 + c\left(\frac{N^2}{2}\right) = \left(4b + \frac{c}{2}\right)N^2 \text{ bit} \quad (6)$$

where b is number of bits needed to store each one of the (i, j) pixel coordinates in LSP or LIP, and c is number of bits needed to store each one of the (i, j) pixel coordinates in LIS. So, $b = log_2 N$ bits, and $c = log_2 N/2$ bits. The LHS-SPIHT uses fixed-size memory of an average of 4 bits/pixel with total memory = 4(N×N) bits. Finally, the MSLS (and the HS-MSLS) also uses fixed-size memory, where the total number of entries of LRS = N×N /4, and it uses 2 bits/pixel, and 1 bit/root, so the total memory of MSLS is equal to:

$$MEM_{MSLS} = 2N^2 + \frac{N^2}{4} + 2c\left(\frac{N^2}{4}\right) = \left(2 + \frac{1}{4} + \frac{c}{2}\right)N^2 \text{ bit} \quad \textbf{(7)}$$

Table 8 depicts the memory requirement of these algorithms for different image sizes. The column (%) represents the percentage of the total memory to the memory required to store the DWT image, which is equal to 16(N×N) bit, as each wavelet coefficient is represented by 16 bits. As shown, the memory of our HS-MSLS is greatly lower than that of SPIHT, and it slightly higher than that of LHS-SPIHT. However, as mentioned previously, the LHS-SPIHT algorithm utilizes the linear indexing technique that maps the DWT image to 1D array, which demands storing both in the memory. So, the image size should be added to the actual memory consumption of LHS-SPIHT.

Table 8: Memory requirements of SPIHT, LHS-SPIHT, and the proposed HS-MSLS for different image sizes

Image size	Memory (KB)					
	SPIHT		LHS- SPIHT		HS-MSLS	
	MEM (KB)	%	MEM (KB)	%	MEM (KB)	%
256×256	284	2.21	32	0.25	46	0.36
512×512	1280	2.50	128	0.25	200	0.39
1024×1024	5696	2.78	512	0.25	864	0.42

V. Conclusion

The paper presented the MSLS and the HS-MSLS algorithms. The MSLS algorithm produces a quality scalable bitstream. As demonstrated from the simulation, MSLS has better PSNR performance, and has lower complexity than its predecessor the SLS algorithm. The proposed HS-MSLS algorithm upgraded the MSLS algorithm to produce a highly scalable bitstream that owns the quality and resolution scalabilities. As such, the image can be easily reconstructed at multiple qualities and resolutions using a simple bitstream parsing process. As shown, this is realized by arranging and identifying the data in each coding pass according to the resolution levels it belongs. As given from the simulation results, the HS-MSLS algorithm has improved PSNR, and runs faster than the other HSIC algorithms. The proposed HS-MSLS is therefore very suitable for sending images over the Internet where the users to be serviced according to their capabilities and desires. Additionally, the high speed and reduced memory advantages of HS-MSLS makes it very appropriate as a part of the real-time scalable video transmission systems [24], and for compressing super-resolution and 3-D images [25].

Declarations

Ethical Approval
Not applicable.

Competing interests
Not applicable.

Authors' contributions
The 2nd author prepared all the tables in the paper.
The 3rd author prepared all the figures in the paper.
All authors revised the paper.

Funding
Not applicable.

Availability of data and materials
The datasets used are available freely on the Internet, and can be accessed using the following link
https://ccia.ugr.es/cvg/CG/base.htm

References

[1] Uthayakumar J., et al., A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications, Journal of King Saud University - Computer and Information Sciences, Vol. 33, No. 2, pp. 119-140, 2021. https://doi.org/10.1016/j.jksuci.2018.05.006
[2] Rohit M. et al., Hybrid and Advanced Compression Techniques for Medical Images, Springer Nature press, 1st edition, 2019, ch2. https://doi.org/10.1007/978-3-030-12575-2
[3] Taubman, D. et al., Embedded block coding in JPEG 2000. Signal Processing: Image Communication, Vol. 17, No. 1, pp. 49-72, 2002. https://doi.org/10.1016/S0923-5965(01)00028-5
[4] Rüefenacht, D., et al., Base-Anchored Model for Highly Scalable and Accessible Compression of Multiview Imagery. *IEEE Transactions on Image Processing,* Vol. 28, No. 7, pp. 3205-3218, 2019. https://doi.org/10.1109/TIP.2019.2894968

[5] Patrick J. V. F., Discrete Wavelet Transformations: An Elementary Approach with Application, Wiley, 2019. https://doi.org/10.1002/9781119555414.ch5
[6] Vetterli M. et al., Wavelets and Subband Coding, Prentice-Hall, New Jersey, 1st edition, 1995.
[7] Said A. et al., 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, 1996. https://doi.org/10.1109/76.499834
[8] Lee, RC., et al., New Modified SPIHT Algorithm for Data Compression System. *J. Med. Biol. Eng.* Vol. 39, pp. 18–26, 2019. https://doi.org/10.1007/s40846-018-0384-z
[9] Ekram K., et al., An efficient and scalable low bit-rate video coding with virtual SPIHT, Signal Processing: Image Communication, Vol. 19, No. 3, pp. 267-283, 2004. https://doi.org/10.1016/j.image.2003.08.019
[10] Ranjan K. Senapati, et. al., Listless block-tree set partitioning algorithm for very low bit rate embedded image compression, AEU - International Journal of Electronics and Communications, Vol. 66, No. 12, pp. 985-995, 2012. https://doi.org/10.1016/j.aeue.2012.05.001
[11] Drozdek, A., Data structure and Algorithms in C++, CENGAGE LEARNING press, 4th ed., 2012, ch3.
[12] Chew, L. W., et al., Reduced Memory SPIHT Coding Using Wavelet Transform with Post-Processing, 2009 International Conference on Intelligent Human-Machine Systems and Cybernetics, Hangzhou, Zhejiang, 2009, pp. 371-374. https://doi.org/10.1109/IHMSC.2009.101
[13] Deepthi, S. A. et al., Image transmission and compression techniques using SPIHT and EZW in WSN," 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, 2018, pp. 1146-1149. http://dx.doi.org/10.1109/ICISC.2018.8398984
[14] Alam M, et al., Modified Listless Set Partitioning in Hierarchical Trees (MLS) For Memory Constrained Image Coding Applications, Current Trends in Signal Processing Vol. 2, No. 2, pp. 56-66, 2012. https://doi.org/10.37591/ctsp.v2i1-3.5124
[15] Al-Janabi, A.K, et al., An efficient and Highly Scalable Listless SPIHT Image Compression Framework, *Journal of Applied Research and Technology*, Vol. 20, No. 2, pp. 173-187, 2022. https://doi.org/10.22201/icat.24486736e.2022.20.2.1269

[16] Y. Meraj et al., Modified ZM-SPECK: A Low Complexity and Low Memory Wavelet Image Coder for VS/IoT Nodes, 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), 2021, pp. 494-500. https://doi.org/10.1109/ESCI50559.2021.9396834

[17] Al-Janabi, A. K, Low Memory Set-Partitioning in Hierarchical Trees Image Compression Algorithm, International Journal of Video & Image Processing and Network Security IJVIPNS-IJENS, Vol. 13, No. 2, pp. 12-18, 2013.

[18] Monauwer, A. et al., 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, No. 3, pp. 36-48, 2012. http://www.tjprc.org/view_paper.php?id=729

[19] Danyali, H. et. al., A., Flexible, highly scalable, object-based wavelet image compression algorithm for network applications, IEE Proceedings - Vision, Image and Signal Processing, Vol. 151, No. 6, pp. 498-510, 2004. https://doi.org/10.1049/ip-vis:20040734

[20] Calderbank, A. et al., Wavelet transforms that map integers to integers, Applied and computational harmonic analysis Vol. 5, No. 3, pp. 332-369, 1998.

[21] Hu, Yueyu, et al., Towards coding for human and machine vision: A scalable image coding approach, in 2020 IEEE International Conference on Multimedia and Expo (ICME), pp. 1-6. IEEE, 2020.

[22] http://www.spiht.com/spiht3.html#mat-spiht (Last visited on Sunday 22 August 2022).

[23] Sara, U., et al., Image Quality Assessment through FSIM, SSIM, MSE and PSNR−A Comparative Study. Journal of Computer and Communications, Vol. 7, pp. 8-18, 2019. https://doi.org/10.4236/jcc.2019.73002

[24] Wu, D. Zhang, et al., Multiview Video Coding Based on Wavelet Pyramids, 2013 International Conference on Computational and Information Sciences, Shiyang, 2013, pp. 225-228. https://doi.org/10.1109/ICCIS.2013.67

[25] Song, Xiaoying et. al., Three-dimensional separate descendant-based SPIHT algorithm for fast compression of high-resolution medical image sequences, IET Image Processing, Vol. 11, No. 1, pp. 80-87, 2017. https://doi.org/10.1049/iet-ipr.2016.0564