Hierarchical Dynamic Range Coding of Wavelet Subbands for Fast and Efficient Image Compression

CIPR Technical Report TR-2007-4

Yushin Cho and William A. Pearlman

March 2007

Center for
Image Processing Research

Rensselaer Polytechnic Institute
Troy, New York 12180-3590
http://www.cipr.rpi.edu

Hierarchical Dynamic Range Coding of Wavelet Subbands for Fast and Efficient Image Decompression

Yushin Cho and William A. Pearlman

Abstract

An image coding algorithm, PROGRES (Progressive Resolution Coding), for a high speed resolution scalable decoding is proposed. The algorithm is designed based on a prediction of the decaying dynamic ranges of wavelet subbands. Most interestingly, because of the syntactic relationship between two coders, the proposed method costs very similar amount of bits as used by uncoded (i.e. not entropy coded) SPIHT. The algorithm bypasses bit-plane coding and complicated list processing of SPIHT in order to obtain a considerable speed improvement, giving up quality scalability, but without compromising coding efficiency. Since each tree of coefficients is separately coded, where the root of the tree corresponds to the coefficient in LL subband, the algorithm is easily extensible to random access decoding.

The algorithm is designed and implemented for both 2D and 3D wavelet subbands. Experiments show that the decoding speeds of proposed coding model are four times and nine times faster than uncoded 2D-SPIHT and 3D-SPIHT respectively, with almost the same decoded quality. The higher decoding speed gain in a larger image source validates the suitability of the proposed method to a very large scale image encoding and decoding.

In the Appendix, we explain the syntactic relationship of the proposed PROGRES method to uncoded SPIHT, and demonstrate that in the lossless case the bits sent to the codestream for each algorithm are identical, except that they are sent in different order.

Y. Cho is with SONY Electronics, Inc., San Jose, CA 95112; E-mail: cho.yushin at gmail.com

W. A. Pearlman is with Rensselaer Polytechnic Institute, Electrical, Computer and Systems Dept., Troy, NY 12180-3590; E-mail: pearlw@ecse.rpi.edu.

This work was carried out in the Center for Image Processing Research, Rensselaer Polytechnic Institute, Troy, NY, USA and was supported in part by the Office of Naval Research under Grant No. N00014-05-1-0507.

I. Introduction

Modern image coding methods [1], like JPEG2000's EBCOT [2], are able to support simultaneous sub-image decompression (ROI), and also quality (SNR), resolution, and spectral scalability [3] [4]. Unfortunately, while the loss in compression incurred by supporting these features can be quite small, they may increase computational complexity significantly.

Quality or rate scalability is commonly done via bit-plane coding, which normally requires several passes through all the pixels or transform coefficients and uses adaptive entropy coding with powerful contexts. However, in many important applications, such as the digital camera, the images need to have a pre-defined quality and any extra effort required for quality scalability is wasted. Furthermore, there are increasingly common applications where one may not be able to afford the additional time and computation complexity required for rate scalability.

Image datasets for scientific and medical applications are growing to enormous sizes in the multi-gigabyte and even terabyte range. These datasets are typically gathered by computer tomography and electron microscopy and are often three-dimensional and even four-dimensional (three-dimensional versus time). Fast decoding of regions of interest in multiple scales of resolution are essential properties for the efficient utilization and exploitation of these data. These properties are needed for rapid browsing in remote sensing and GIS applications, for example. In most cases, scientific image data must be encoded losslessly and selected portions of this data must be decoded losslessly, so as to guarantee no loss of analytic or diagnostic accuracy. In other cases, a designated high quality is often sufficient for the application. Therefore, the property of rate scalability is never utilized and is a considerable impediment to fast decoding.

In this paper, we consider fast, efficient coding that supports resolution scalability and efficient decompression of sub-images by random access decoding. The methods are implemented both for two-dimensional and three-dimensional images, but for the sake of clarity, we shall explain our algorithm in the context of two-dimensional images, as the extension to three dimensions then becomes obvious. Our solution addresses the challenge of avoiding compression loss, while simultaneously reducing complexity by using neither bit-plane coding (and its contexts) nor subsequent entropy coding.

The original EZW [5] and SPIHT [6] do not support resolution scalability, since they do not code the resolution boundaries. They also do not support random access decoding [3] [4], i.e., decoding of a given region-of-interest by random access to the compressed bitstream.

The algorithm to be presented, PROGRES (Progressive Resolution Coding) is a method that exploits

the same image properties as SPIHT, but is adapted to support resolution scalability and random access decoding with great encoding and decoding speed. For a pre-defined quality, it can efficiently decompress any designated image region at several resolutions.

This paper is organized as follows. Section II provides an overview of the proposed coding method. In Section III, we illustrate how dynamic ranges of coefficients are hierarchically and efficiently coded in each wavelet tree, where the decrease of dynamic range is shared among a group of coefficients. Then a description of the coding algorithm is given in Section IV. A step-by-step description of the PROGRES coding scheme is given in Section V. Section VI presents the experimental results for both 2D and 3D image sources, where the coding efficiency and coding speed of PROGRES are compared to the uncoded SPIHT algorithm. Section VII concludes the paper. In the Appendix, the relationship between PROGRES and SPIHT is explained in detail.

II. Previous Work and Overview

Speed improvements were observed in hybrid forms of bit-plane coding, where once an image transform coefficient is classified as significant during a bit-plane pass, its sign and all its less significant bits are encoded together, so that refinement passes are not needed [7].

A simple hierarchical coder, SWEET, proposed by Andrew [8], codes the coefficients with multiple bit planes based on both hierarchical set partitioning and block-coding. This set partitioning scheme does not achieve the performance of SPIHT [6], but the coefficient coding scheme (with a bin of$n$ bits) does lead to reduction in computations.

The low-complexity, embedded wavelet based coder presented by Ordentlich et. al. [9] uses neither zerotree nor arithmetic coding and instead encodes bit lanes with faster adaptive Elementary Golomb codes. This paper shows similar PSNR performance to SPIHT-AC (Arithmetic Coded) and superior to SPIHT (uncoded). No execution time numbers were reported and the software is no longer available [10]. Berghorn *et al.* [11] also presented a fast embedded wavelet-based coding algorithm, based on adaptive arithmetic coding of significance state symbols and distance differences of significant2×2 blocks in scan modes across scales. Adaptive arithmetic coding is also used for coding the bit planes in the refinement passes. Total execution times, including wavelet transform, proved to be smaller than SPIHT-AC. Both Ordentlich *et al.* [9] and Berghorn *et al.* [11] rely on adaptive entropy coding of bit planes in order to produce embedded codestreams. Both these methods require multiple passes through the wavelet coefficients and have no option for bypassing bit plane coding to gain coding and decoding speed.

Oliver and Malumbres [12] presented LTW (Lower-Tree Wavelet), which is another solution for

resolution scalable wavelet image coding with low complexity, based on non-embedded coding. The lower-trees are equivalent to the zerotrees of pre-quantized wavelet coefficients, where the quantization step size is 2^{rplane} ($rplane$ is the number of the lowest bit-planes to drop). Arithmetic coded symbols for zerotree, isolated zero, and number of bits to represent the magnitude of coefficients are used.

Highly scalable SPIHT (HS-SPIHT), presented by Danyali and Mertins [13], enables the SPIHT algorithm to have spatial scalability. It introduced multiple resolution-dependent lists and a resolution-dependent sorting pass. Thereby, it can keep the important features of the original SPIHT algorithm such as compression efficiency, full SNR scalability and low complexity. However, the focus of this method is on the expansion of scalability, instead of reduction of computation. This method also does not allow random access decoding.

Similar to other wavelet based image coding methods [5] [6] [14] [15] [16] using intra and inter-band coding contexts, our method is based on two properties of natural images: (a) energy in each subband normally decreases with spatial frequency [17]; and (b) statistics in a local neighborhood are similar. Thus, we also use the strategy of coding wavelet coefficients following the order of expected importance from most to least significant bits, from low to high-resolution subbands. However, to reduce the computational burden we do not follow a bit plane-by-bit plane scan. Each coefficient, represented by sign and magnitude, is processed only once.

The wavelet transform of an image is partitioned into a number of non-overlapping spatial orientation trees, each rooted by a coefficient in the lowest spatial frequency subband. This enables an easy extension of the proposed coding scheme to random access decoding, because each tree represents a region of the image geometrically similar to the root in the lowest spatial frequency subband.

The dynamic range of a tree of wavelet coefficients is represented as a*dynamic range number*, which gives the number of bits required to represent every magnitude of the coefficient in the tree. Based on the assumption of decaying power spectral density, the dynamic ranges of descendant subtrees are highly likely to be smaller than that of the parent tree. We code the amount of this decrease in dynamic range number by lossless differential coding.

In addition, since a local neighborhood of wavelet coefficients has similar statistics, the descendant subtrees can share the information of decrease in dynamic range. This procedure is hierarchically applied to each coefficient resolution by resolution. Because a decrease in dynamic range between parent and children coefficients affects not just those children (i.e. the second generation) but all their descendants also, the presented dynamic range coding method efficiently represents the hierarchy of dynamic ranges over a spatial orientation tree.

III. Hierarchical Dynamic Range Coding

A. Dynamic Range of Coefficients and a Tree of Coefficients

We use $c_{i,j}$ and $s_{i,j}$ to represent, respectively, a wavelet coefficient at location (i,j), and a tree of coefficients with root at location (i,j) (See Fig. 1).

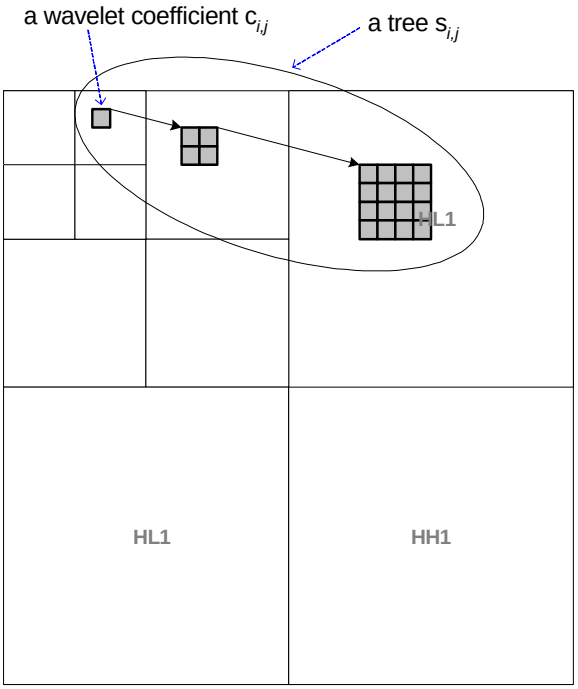

Fig. 1. Wavelet coefficient $c_{i,j}$ and a tree of coefficients $s_{i,j}$

As mentioned above, to represent the magnitude compactly, the number of required bits should be known in advance. When the *dynamic range* of a coefficient magnitude is represented by the number of bits, k, the magnitude varies in the range of $[0, 1, \cdots, 2^k - 1]$. We shall call that number of bits the *dynamic range number*, which is analogous to the set number in AGP described in [18]. For a non-zero coefficient, an additional bit is required to represent its sign.

Each tree (a spatial orientation tree) will contain a different dynamic range of magnitudes, based on the activity of its coefficients. We define the *dynamic range number* $r_{i,j}$ of the tree $s_{i,j}$ as:

$$r_{i,j} = \lceil \log_2(\max_{c_{p,q} \in s_{i,j}} |c_{p,q}| + 1) \rceil \,,$$

which accounts for how many bits are required to represent every coefficient magnitude in the tree.

B. Coding of Dynamic Ranges in a Tree: One-Stage Prediction

When a tree is partitioned into its subtrees, each subtree will have a different dynamic range, probably a decreased one, because the root coefficient of the tree is likely to have the largest magnitude in the tree. Thus, subtrees (i.e. child trees) are likely to have smaller dynamic ranges than that of their parent tree.

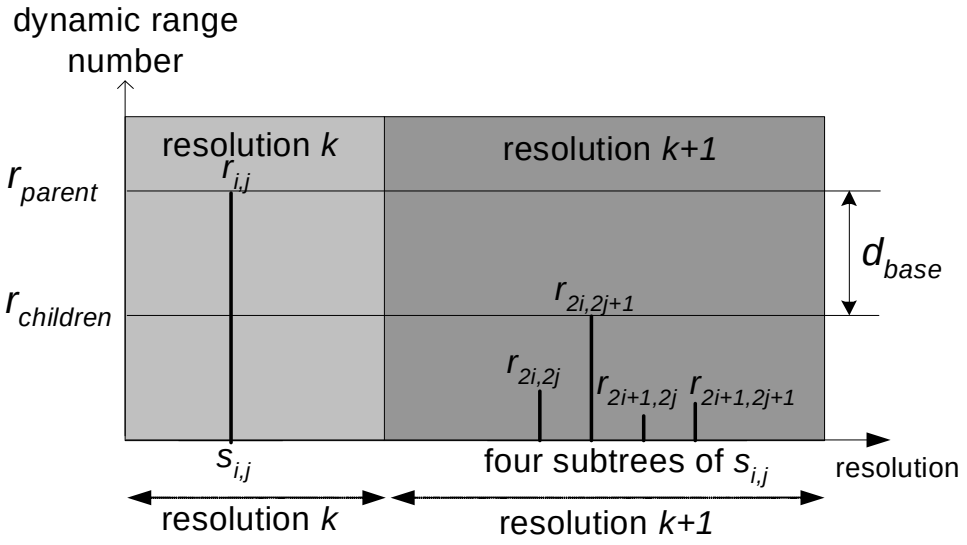

Fig. 2. One-stage dynamic range coding : each subtree $s_{m,n}$ of $s_{i,j}$ is coded by using the same dynamic range number of $r_{children}$, which is reconstructed by : $r_{children} = r_{parent} - d_{base}$. The information d_{base} is actually coded to represent the coefficients in subtree $s_{m,n}$.

Therefore, it is a good idea to predict the dynamic range of each subtree based on the dynamic range of a parent tree, as shown in Fig. 2. Assuming that a parent tree $s_{i,j}$ is partitioned into four subtrees, $s_{2i,2j}$, $s_{2i,2j+1}$, $s_{2i+1,2j}$, and $s_{2i+1,2j+1}$, then the $r_{2i,2j}$, $r_{2i,2j+1}$, $r_{2i+1,2j}$, and $r_{2i+1,2j+1}$ are the dynamic ranges for each subtree, respectively.

Let $I(i,j) = \{(2i,2j),(2i,2j+1),(2i+1,2j),(2i+1,2j+1)\}$ denote the set of position indices of the four subtrees of tree $s_{i,j}$. Then, the range, $r_{m,n}$, is defined for each subtree $s_{m,n}$, $(m,n) \in I(i,j)$.

Now, for representing the dynamic range number of each subtree $s_{m,n}$, we encode

$$d_{base} \quad = \quad r_{parent} - r_{children}\,,$$

where r_{parent} is the dynamic range number $r_{i,j}$ of the parent tree $s_{i,j}$ and $r_{children}$ is the dynamic range

number of the subtrees of $s_{i,j}$, i.e.

$$r_{children} = \max_{(m,n)\in I(i,j)} (r_{m,n}) .$$

Note that one dynamic range number, $r_{children}$, is used to represent the magnitudes in all subtrees.

Then, in the decoder side, given the information of r_{parent} and d_{base}, $r_{children}$ can be reconstructed and we use this value as the dynamic range number for the subtrees $s_{m,n}$. Note that the information of $r_{parent} - d_{base}$ is common to every subtree $s_{m,n}$.

Now, the coded information for the tree $s_{i,j}$ with two resolution scales will be:

$$r_{i,j}, c_{i,j}, d_{base}, c_{2i,2j}, c_{2i,2j+1}, c_{2i+1,2j}, c_{2i+1,2j+1},$$

where $c_{2i,2j}, c_{2i,2j+1}, c_{2i+1,2j}, c_{2i+1,2j+1}$ are root coefficients of each subtree. The $c_{i,j}$ and $c_{m,n}$, $(m,n) \in I(i,j)$ contain sign information. The $c_{i,j}$ comprises $r_{i,j}+1$ bits of information including a sign. The $c_{m,n}$, $(m,n) \in I(i,j)$, comprise $r_{i,j} - d_{base} + 1$ bits of information including a sign.

There is a reason why we choose d_{base} rather than $r_{children}$ to code, where $r_{children} = r_{parent} - d_{base}$. From our experience, it is more probable that $d_{base} \leq r_{children}$, in other words, it can be observed that the probability $P(d_{base} \leq r_{children}) > 0.5$ in any wavelet transformed image and $P(d_{base} \leq r_{children})$ is getting closer to 1 for lower bit rates. This explains why coding d_{base} will cost fewer bits than coding $r_{children}$.

The above coding scheme of dynamic range is applied to every two adjacent resolution scales,k and (k+1), $k = 0$ to $M-2$, where $M-1$ is the highest resolution. In this case, note that the number of parent-children relationships increases four times for each additional resolution scale of a 2D transform image.

By coding the decrease in dynamic range number or d_{base} for each group of four subtrees, the amount of bit savings is simply $3 \times d_{base}$ bits since we would need a different decrease in the dynamic range number for each subtree if we did not use the common d_{base} information.

C. Coding of Dynamic Ranges in a Tree: Two-Stage Prediction

The PROGRES image coder is built on an extension of the idea of dynamic range coding (by a one-stage prediction) described in the previous subsection III-B. Instead of sharing the d_{base} value among four children coefficients, it is shared by sixteen coefficients in practice, whose parents are at the same tree level. In other words, these sixteen children have the same grandparent, as shown in Fig. 3. By doing this, about 1.5% coding gain is achieved in our experiments. However, sharing a 'decrease in dynamic range' with more than sixteen children does not seem to give any improvements in coding gain.

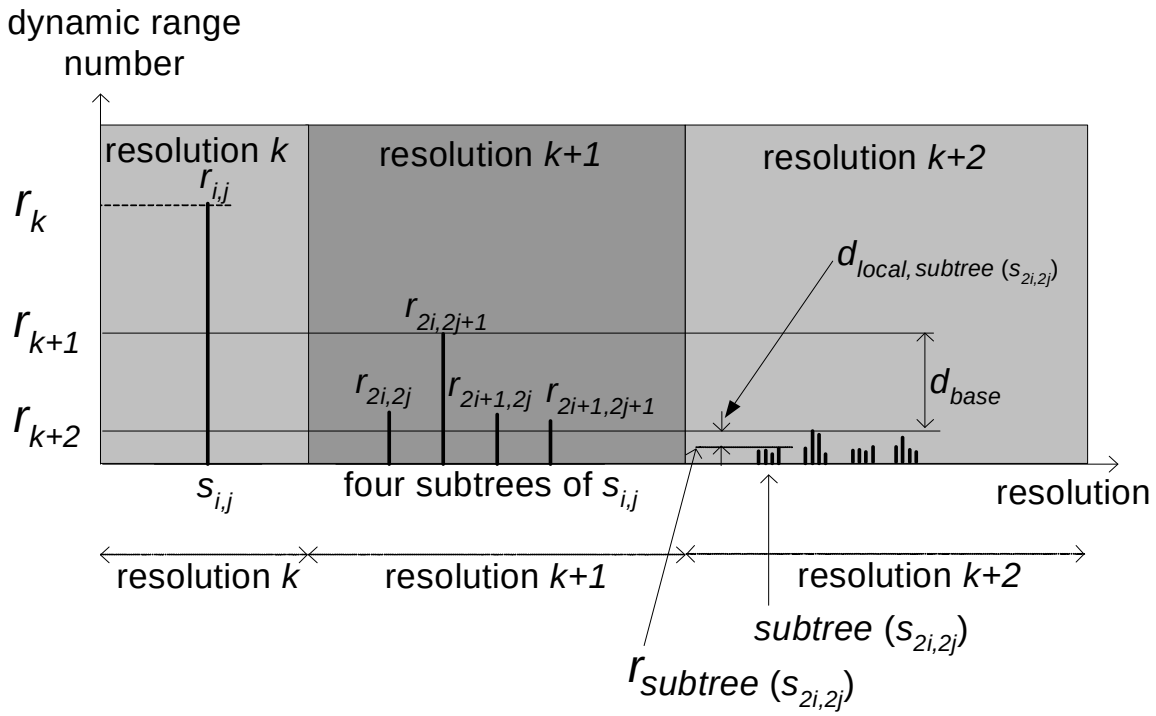

Fig. 3. Two-stage dynamic range coding : the dynamic range for each tree in $subtree(s_{m,n})$, is reconstructed by: $r_{k+1} - d_{base} - d_{local,\, subtree(s_{m,n})}$, where the information of $r_{k+2} = r_{k+1} - d_{base}$ is common to every tree $subtree(s_{m,n})$. (Here, an example for $(m,n) = (2i, 2j)$ is shown.) The r_k, r_{k+1}, r_{k+2} stand for the dynamic range numbers of trees rooted at resolution k, k+1, k+2, respectively. The $r_{subtree(s_{2i,2j})}$ is the dynamic range number of all subtrees of $s_{2i,2j}$.

We assume $(m,n) \in I(i,j)$ as before. Then, in Fig. 3, $subtree(s_{m,n})$ at resolution k+2 indicates the subtrees of each tree $s_{m,n}$ at resolution k+1. Our goal here is to code the coefficients of $subtree(s_{m,n})$ located at resolution k+2, i.e. the grandchildren coefficients of the tree $s_{i,j}$.

In Fig. 3, the r_k, r_{k+1}, r_{k+2} stand for the dynamic range numbers of trees rooted at resolutions k, k+1, k+2, respectively. The information of r_{k+1} is available to every $subtree(s_{m,n})$ at resolution $k+2$, since every coefficient $c_{m,n}$ at the resolution k+1 (which is the root of $subtree(s_{m,n})$) is coded with r_{k+1} bits. Now, the dynamic range for each $subtree(s_{m,n})$ at resolution k+2 can be predicted in two stages. First, the r_{k+2} is predicted by d_{base} from r_{k+1}, and then the $d_{local,subtree(s_{m,n})}$ is further used to predict the dynamic range for each $subtree(s_{m,n})$.

Thus, all trees in $subtree(s_{m,n})$ has the dynamic range number, $r_{k+1} - d_{base} - d_{local,\, subtree(s_{m,n})}$, where $r_{k+1} - d_{base} = r_{k+2}$ and $r_{k+2} - d_{local,\, subtree(s_{m,n})} = r_{subtree(s_{m,n})}$. As a result, $subtree(s_{m,n})$'s sixteen coefficients at resolution k+2 are sharing the information of d_{base}, which enables the PROGRES

algorithm to code the dynamic ranges efficiently.

IV. Coding Algorithm

A. Algorithm Description

The description of the coding algorithm, which we call PROGRES, is given in Table 4 and the explanation of the variables used in the algorithm is provided in Fig. I. Assume we have M resolutions, $0, 1, 2, \ldots, M-1$, 0 for the lowest resolution, and $M-1$ for the highest resolution. The algorithm is repeatedly applied to each wavelet tree separately. At the beginning of the algorithm, the list L will always contain the root coordinate of the tree yet to be coded. The algorithm then encodes each tree of wavelet coefficients successivly from low to high spatial resolution. Once the list L is initialized, the root coefficient magnitude is coded with MAX_R bits, which is a predetermined maximum dynamic range of coefficients. Then, each of the three children of a root coefficient, in HL, LH, and HH subbands, is coded using the decrease of dynamic range bits.

For each resolution k (Statement 10 of the algorithm), first the difference (i.e. the d_{base}) of two dynamic range numbers between resolution k+1 and k+2, i.e. r_{k+1} and r_{k+2} respectively, is calculated and coded. And then, based on the dynamic range number for resolution $k+2$, i.e. r_{k+2}, the decrease (i.e. the $d_{local,\, subtree(s_{m,n})}$) of dynamic range number to each $subtree(s_{m,n})$'s four coefficients at resolution k+2 is calculated and coded. Finally, each $subtree(s_{m,n})$'s four coefficients at resolution k+2 is coded with $r_{k+1} - d_{base} - d_{local,\, subtree(s_{m,n})}$ bits.

While d_{base} represents the difference of dynamic range numbers between resolution k+1 and k+2, the $d_{local\, subtree(s_{m,n})}$ represents the decrease of dynamic range numbers between resolution k+2 and $r_{subtree(s_{m,n})}$. The statement $d_{base} \leftarrow r_{k+1} - r_{k+2}$ at 10.(a).iii and the statement $d_{local} \leftarrow r_{k+2} - r_{subtree}$ at 10.(a).iv.B correspond to these range numbers, respectively.

B. Unary Coding

Unary coding is used for coding the decreases in dynamic ranges over two adjacent resolutions, i.e. d_{base} and d_{local}. Unary coding is a prefix code that is nearly[1] the optimal Huffman code for the exponential probability distribution. The unary code for an integer number is that number of 1's followed by a single 0. For example, $0, 10, 110, \ldots$ represent the codewords for the events $x_0, x_1, x_2, \ldots$ with the probabilities $p(x_0) = \frac{1}{2}$, $p(x_1) = \frac{1}{4}$, $p(x_2) = \frac{1}{2^3}, \ldots$, respectively. When the distribution of the source follows the exponential probability distribution, the average coding rate of unary coding is close to 2 bits/symbol.

[1]Except for the two least probable symbols, which have the same codeword length and differ by 1 bit for the Huffman code.

1) Initialize a list $L \leftarrow$ a tree rooted at the lowest resolution (i.e. LL subband);
2) Find the maximum dynamic range number r_0;
3) **if** $r_0 = 0$ **return**; // no coefficients to encode ?
4) Binary encode the magnitude of root coefficient with MAX_R bits and encode its sign bit with one bit (MAX_R is a predefined number);
5) $r_1 \leftarrow$ a maximum dynamic range number of the subtrees of the root coefficient;
6) $d_{base} \leftarrow r_0 - r_1$;
7) Unary encode d_{base};
8) **if** $r_1 = 0$ **exit** // means, nothing to encode (i.e. zerotree), thus exit
9) Binary encode the coefficient magnitudes at resolution 1 using r_1 bits and encode their sign bits;
10) **for** each resolution level $k = 0$ to $M - 3$
    a) **for** each tree j rooted at current resolution level k
        i) $r_{k+1} \leftarrow$ a maximum dynamic range number of *subtree*(tree j);
        ii) $r_{k+2} \leftarrow$ a maximum dynamic range number of *subtree*(*subtree*(tree j));
        iii) $d_{base} \leftarrow r_{k+1} - r_{k+2}$;
        iv) Unary encode d_{base};
        v) **if** $r_{k+1} = 0$ **continue**; // Nothing to encode for tree j? Then, goto a)
        vi) **for** each subtree i (i.e. rooted at resolution k+1)
            A) $r_{subtree} \leftarrow$ maximum dynamic range number of *subtree*(tree i);
            B) $d_{local} \leftarrow r_{k+2} - r_{subtree}$;
            C) Unary encode d_{local};
            D) **if** $r_{subtree} = 0$ **continue**; // Nothing to encode for tree i? Then, goto vi)
            E) Binary encode the children coefficients (i.e. at resolution k+2) of subtree i using $r_{subtree}$ bits for each and encode their sign bits;
            F) Append subtree i to the list L for next resolution coding;
        vii) Remove the current tree j from the list L;

Fig. 4. PROGRES Coding Algorithm (Explanation of the variables used in the algorithm is provided in Table I).

L	A list of coordinates of roots of trees yet to be coded.
M	Number of spatial resolutions numbered $k = 0, 1, 2, \ldots, M-1$; $k = 0$ is LL subband.
r_0, r_1	Dynamic ranges of a input tree with resolutions $M-1$ to 0 and $M-1$ to 1, respectively.
MAX_R	A predetermined number larger than the maximum dynamic range of the input wavelet tree.
d_{base}	Difference of dynamic range between two adjacent resolutions.
r_{k+1}, r_{k+2}	Variables to store the dynamic ranges shown in Fig 3.
subtree(tree j)	A set of all subtrees of tree j
$r_{subtree}$	Dynamic range of all subtrees of a tree rooted at resolution k+1.
d_{local}	Difference of dynamic range between r_{k+2} and $r_{subtree}$.

TABLE I

VARIABLE DESCRIPTION OF PROGRES CODING ALGORITHM IN FIG. 4.

C. Bitstream structure

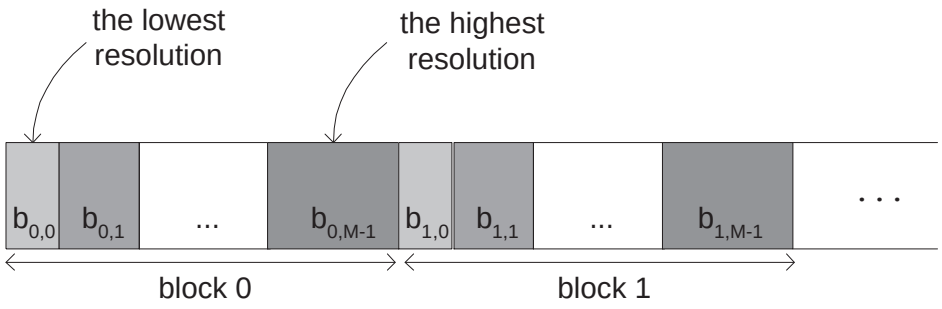

Fig. 5. A bitstream structure ($b_{i,j}$ notates the resolution j of the subimage i)

For M levels of wavelet decomposition, the number of coefficients in each tree is $2^M \times 2^M$, which are the dimensions of the corresponding subimage of the source image located at the root coordinates. (We call this 'subimage' or 'image block' here.) The structure of the output bitstream for a given image block i is a sequence of code segments $b(i, j)$ for resolutions $j = 0$ through $j = M-1$, as shown in Fig. 5. Whenever a full bitstream block i is decoded, the full resolution decoded subimage i is obtained. Decoding only segments $b(i, 0)$ through $b(i, K-1)$ produces a reconstructed subimage reduced in resolution by 2^{M-K}. Therefore, we see that the PROGRES coder is truly resolution progressive in encoding and decoding.

Each tree, rooted at each coefficient in LL subband, is encoded independently of other trees. Random

access decoding is made possible from this property. At the beginning of the part of the bitstream for each coded tree (i.e. each bitstream block in Fig. 5), header information of the coded tree size can be simply added during encoding. In this way, each tree of wavelet coefficients can be randomly accessed in the bitstream and decoded individually. In addition, the progressive resolution decoding works together with random access decoding. This means that once we designate coordinates of the subimage to be decoded, it can be decoded from lower to higher resolutions progressively. Fig. 6 illustrates this idea.

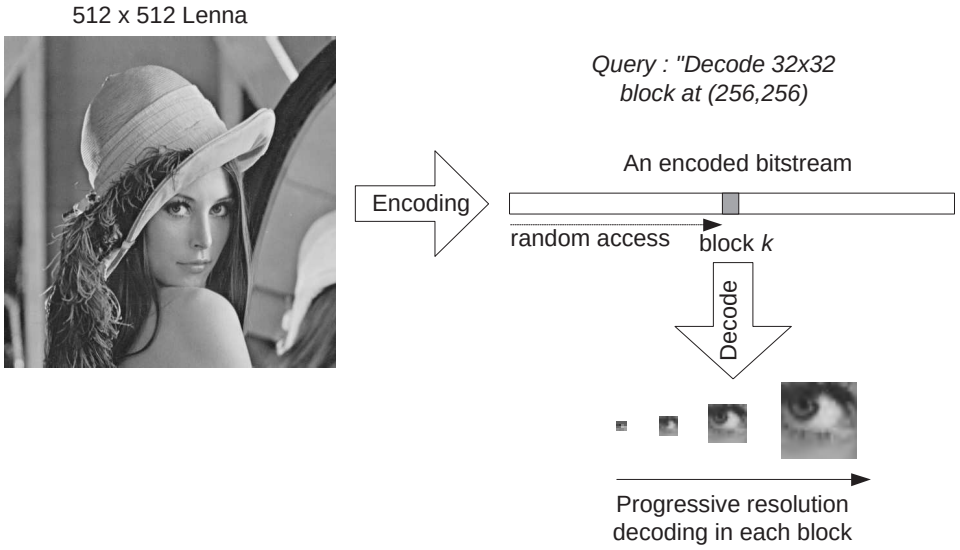

Fig. 6. Simultaneously progressive resolution and random access decoding

V. An Example of PROGRES Coding

For the wavelet coefficients of Fig. 7, the step by step demonstration of PROGRES coding is described in Tables II and III. Each row of the table shows each coding step sequentially.

The basic processing order of the source wavelet coefficients is resolution by resolution. And for each resolution, the coefficient is visited by the numbering policy shown in Fig. 8. Note that this policy is the same as the BFS (Breadth First Search) algorithm. The coefficients in the next higher level resolution will never be processed until the ones in the current resolution level are all finished. In the last column of the Tables II and III, a pair of parenthesized number and a number such as (0) 96 and (1) -6 indicates the scanning order of current wavelet coefficient and the wavelet coefficient itself (i.e. the coefficient 96 is processed first and the coefficient -6 is processed in the second).

First, the initial dynamic range number $r_{parents}$ for the 16×16 wavelet coefficients block (representing the 16×16 image block) is 7. This means that the maximum coefficient magnitude can be $2^7 - 1$, which is 127. The coefficient range with sign is $[-127, 127]$. All the coefficient magnitudes in this block can be represented by 7 bits, although the dynamic range prediction scheme of PROGRES will further reduce the dynamic range through increasing resolutions. The information about the number of bits required to represent the dynamic range can be viewed as the set number in the AGP method discussed in [18]. The actual maximum coefficient is 96 as seen in Fig. 7, which is located in the LL subband, i.e. the resolution level 0. Thus, the magnitude 96 and its sign '+' are coded by 7 + 1 bits.

Now, each of the three coefficients (1) -6, (2) -25, and (3) -8 at resolution level 1 (See Fig. 7) is coded with 5 bits to accommodate the range maximum for '(2) -25'. Note that, the root coefficient (0) has three children coefficients, (1), (2), and (3), which is different from other coefficients that have four children coefficients.

In Tables II and III, note that ,if the 'current dynamic range number' becomes 0, there is nothing to code, since all the coefficients in the group are just zeros.

96	-6	1	-7	1	0	2	-3	0	0	0	0	0	0	0	0
-25	-8	2	-10	0	0	0	0	0	0	0	0	0	0	0	0
-3	-2	0	3	0	3	-1	-7	0	0	0	0	0	0	0	0
3	-2	-3	-4	3	1	0	-6	0	0	0	0	0	0	0	-1
-1	0	-1	1	0	0	0	1	0	-1	0	0	0	1	-1	0
0	0	1	-1	0	0	1	0	0	-1	1	0	0	-1	0	-2
0	1	-1	1	-1	2	-2	-1	0	0	0	0	0	0	0	0
-1	0	-5	-4	0	-2	0	3	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	-1	-1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	-1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0

Fig. 7. 16×16 quantized wavelet transformed image, four levels of decomposition, truncated to integer.

VI. Experimental Results: 2D and 3D Coding

Tests were performed using an Intel 2.0 GHz Xeon processor, MS-Windows 2000, and Visual C++ 6.0 Compiler with speed optimization. The compression and decompression times are measured in CPU cycles for 2D images (since 2D image coding times are very small) and seconds for 3D images (video). The

0	1	4	5	16	17	20	21	64	65	68	69	80	81	84	85
2	3	6	7	18	19	22	23	66	67	70	71	82	83	86	87
8	9	12	13	24	25	28	29	72	73	76	77	88	89	92	93
10	11	14	15	26	27	30	31	74	75	78	79	90	91	94	95
32	33	36	37	48	49	52	53	96	97	100	101	112	113	116	117
34	35	38	39	50	51	54	55	98	99	102	103	114	115	118	119
40	41	44	45	56	57	60	61	104	105	108	109	120	121	124	125
42	43	46	47	58	59	62	63	106	107	110	111	122	123	126	127
128	129	132	133	144	145	148	149	192	193	196	197	208	209	212	213
130	131	134	135	146	147	150	151	194	195	198	199	210	211	214	215
136	137	140	141	152	153	156	157	200	201	204	205	216	217	220	221
138	139	142	143	154	155	158	159	202	203	206	207	218	219	222	223
160	161	164	165	176	177	180	181	224	225	228	229	240	241	244	245
162	163	166	167	178	179	182	183	226	227	230	231	242	243	246	247
168	169	172	173	184	185	188	189	232	233	236	237	248	249	252	253
170	171	174	175	186	187	190	191	234	235	238	239	250	251	254	255

Fig. 8. Coefficients scanning order in PROGRES algorithm for 16×16 image block

2D version of the proposed PROGRES algorithm is straightforwardly extended to a 3D version using three-dimensional DWT and IDWT, three-dimensional wavelet coefficient trees of nominal branching factor 8, and dynamic range coding of 8 children sets and 64 grandchildren sets.

The binary uncoded versions of 2D-SPIHT [6] and 3D-SPIHT [19] from Rensselaer Polytechnic Institute (RPI), which do not use arithmetic coding, are chosen for comparison. Wavelet transformation times are not included, in order to measure speeds of encoding and decoding only. Six and eight levels of wavelet decomposition with Daubechies 9/7 filters [20] are used for Lena and Woman, respectively.

The PROGRES scheme performs lossless coding of quantizer bin numbers of the pre-quantized wavelet transform of the source image. Note that both SPIHT and PROGRES used here do not use subsequent entropy coding of the code streams.

A. 2D Case

The 2D image coding times for 8 bpp gray scale images, 512×512 Lena and 2048×2048 Woman, at the rate of 0.125, 0.25, 0.5 and 1.0 bpp are shown in Table IV.

0.125 298604638 227449803 0.25 430903525 370313769 0.5 666157506 616700070 1.0 1193089359 1066487919

Because PROGRES uses no context-based, adaptive entropy coding, a fairer comparison in PSNR is to binary, uncoded SPIHT, instead of entropy coding based methods such as [12] [9] [11]. For time complexity comparison, we compare with LTW [12] algorithm, providing encoding and decoding times in millions of CPU cycles.

Table IV shows that the encoding speed gains of PROGRES over SPIHT are, 1.02 to 1.95 times for

TABLE II

STEP BY STEP DEMONSTRATION OF PROGRES CODING

(RESOLUTION 0 THROUGH 3. THE PARENTHESIZED NUMBERS INDICATE THE ORDER OF COEFFICIENT ENCODING)

Resolution level	$r_{parents}$	d_{base}	$r_{children}$	d_{local}	Current dynamic range number	Coded information	
						diff	(scan order) coefficient value
0	7				7	7	(0) 96
1	7	2	5		5	2	(1) -6 (2) -25 (3) -8
2	5	1	4		4	1	
2				0	4	0	(4) 1 (5) -7 (6) 2 (7) -10
2				1	3	1	(8) -3 (9) -2 (10) 3 (11) -2
2				1	3	1	(12) 0 (13) 3 (14) -3 (15) -4
3	4	1	3		3	1	
3				2	1	2	(16) 1 (17) 0 (18) 0 (19) 0
3				1	2	1	(20) 2 (21) -3 (22) 0 (23) 0
3				1	2	2	(24) 0 (25) 3 (26) 3 (27) 1
3				0	3	0	(28) -1 (29) -7 (30) 0 (31) -6
3	3	0	3		3	1	
3				2	1	2	(32) -1 (33) 0 (34) 0 (35) 0
3				2	1	2	(36) -1 (37) 1 (38) 1 (39) -1
3				2	1	2	(40) 0 (41) 1 (42) -1 (43) 0
3				0	3	0	(44) -1 (45) 1 (46) -5 (47) -4
3	3	1	2		2	1	
3				2	0	2	
3				1	1	1	(52) 0 (53) 1 (54) 1 (55) 0
3				0	2	0	(56) -1 (57) 2 (58) 0 (59) -2
3				0	2	0	(60) -2 (61) -1 (62) 0 (63) 3

Lena and 1.06 to 2.59 times for Woman. The speed improvement over SPIHT in decoding is four times on average, 2.79 to 4.0 times for Lena and 3.07 to 5.25 times for Woman over the four bit rates in the Table. The difference between encoding and decoding times of PROGRES coder is mostly due to the tree analysis step in the encoder side, which recursively shifts up the maximum coefficient magnitude from bottom to top across the wavelet coefficient trees. Also, it is seen in the Table that PROGRES outperforms the coding speed of LTW in [12], up to two times in encoding and up to seven times in decoding. LTW exploits entropy coding of symbols and gives better coding efficiency than SPIHT and PROGRES (LTW is 0.1 to 0.2 dB better than SPIHT-AC and 0.6 dB better than uncoded SPIHT for the bit rates 0.125, 0.25, 0.5, and 0.1 bpp).

For the bitrates 0.25 to 1.0 bpp of Lena, the encoding algorithm of Berghor*et al.* [11] is approximately 1.6 to 1.5 times faster than SPIHT-AC encoding; and decoding is 1.8 to 0.9 times faster than SPIHT-AC

TABLE III

STEP BY STEP DEMONSTRATION OF PROGRES CODING (RESOLUTION 4)

(CONTINUED FROM TABLE II. THE PARENTHESIZED NUMBERS INDICATE THE ORDER OF COEFFICIENT ENCODING)

Resolution level	$r_{parents}$	d_{base}	$r_{children}$	d_{local}	Current dynamic range number	Coded information	
						diff	(scan order) coefficient value
4	1	1	0		0	1	
4	2	1	1		1	1	
4				1	0	1	
4				1	0	1	
4				1	0	1	
4				0	1	0	(92) 0 (93) 0 (94) 0 (95) -1
4	2	1	1		1	1	
4				0	1	0	(96) 0 (97) -1 (98) 0 (99) -1
4				0	1	0	(100) 0 (101) 0 (102) 1 (103) 0
4				1	0	1	
4				1	0	1	
4	3	1	2		2	1	
4				1	1	1	(112) 0 (113) 1 (114) 0 (115) -1
4				0	2	0	(116) -1 (117) 0 (118) 0 (119) -2
4				2	0	2	
4				2	0	2	
4	1	0	1		1	0	
4				0	1	0	(128) -1 (129) 0 (130) 0 (131) 0
4				1	0	1	
4				1	0	1	
4				1	0	1	
4	1	0	1		1	0	
4				1	0	1	
4				1	0	1	
4				1	0	1	
4				0	1	0	(156) 0 (157) 0 (158) 0 (159) 1
4	1	0	1		1	0	
4	3	2	1		1	2	
4				1	0	1	
4				0	1	0	(180) -1 (181) -1 (182) 1 (183) 0
4				0	1	0	(184) 1 (185) 0 (186) 0 (187) -1
4				0	1	0	(188) -1 (189) 0 (190) 0 (191) 0
4	1	1	0		0	1	
4	2	2	0		0	2	
4	2	2	0		0	2	

TABLE IV

THE COMPARISON OF CODING TIMES BETWEEN PRESENTED METHOD AND OTHER METHODS (2D IMAGE SOURCES : LENA 8 BPP 512×512, WOMAN 8 BPP 2048×2048), WAVELET TRANSFORM TIMES ARE NOT INCLUDED

	Encoding			Decoding		
Bitrate	(cycles $\times 10^6$)			(cycles $\times 10^6$)		
(bpp)	SPIHT	LTW	PROGRES	SPIHT	LTW	PROGRES
Lena						
0.125	24.25	34.7	23.66	4.46	12.3	1.60
0.25	30.87	38.9	26.12	7.78	17.4	2.61
0.5	46.18	46.7	29.01	16.04	27.1	4.55
1.0	67.86	62.4	34.80	33.31	47.1	8.32
Woman						
0.125	400.66	298.60	378.43	73.90	227.44	24.09
0.25	524.05	430.90	404.34	150.11	370.31	41.92
0.5	788.30	666.15	450.13	307.33	616.70	74.71
1.0	1370.54	1193.08	528.42	675.15	1066.48	128.42

TABLE V

DECODED QUALITY IN PSNR: PROGRES VS UNCODED SPIHT, LENA 512 × 512, 8 BPP

Bit rate	2D PROGRES	2D SPIHT
0.125	30.6742	30.7198
0.25	33.7492	33.7245
0.5	36.8877	36.8714
1.0	39.9145	40.0284

decoding. Note that their results include wavelet transform and I/O time, so it is uncertain whether their coding algorithm by itself is faster. (Their PSNR results for Lena are slighter below those of SPIHT-AC.) The relative speed of uncoded SPIHT is known to be up to two times faster than SPIHT-AC, depending on rate [6]. Thus, we estimate that the speed of Berghorn algorithm in [11] is similar to uncoded SPIHT, so must be considerably slower than the proposed PROGRES algorithm.

The main reason that PROGRES decodes (and encodes) faster than SPIHT is that it avoids bitplane

TABLE VI

DECODED QUALITY IN PSNR: PROGRES VS UNCODED SPIHT, WOMAN 2048×2048, 8 BPP

Bit rate	2D PROGRES	2D SPIHT
0.125	26.8938	26.9300
0.25	29.4182	29.4253
0.5	33.0248	32.9183
1.0	37.7560	37.7471

coding. In PROGRES, each coefficient is completely reconstructed by accessing the coefficient only once. However, each coefficient in SPIHT (and many bit-plane coding algorithms) needs multiple passes to fully reconstruct itself. The significant coefficients stored in the LSP are processed by the refinement pass, adding one bit at each bitplane pass below its most significant bitplane. Furthermore, the set partitioning information managed by the LIS list in SPIHT is growing two-dimensionally, i.e. along resolutions and bitplanes. The same is happening to the LIP list, until a coefficient is classified as significant and moved to the LSP. These three list processings account for much heavier computations than the simple BFS (Breadth First Search) traversal used in PROGRES for hierarchical reconstruction of coefficients.

The decoded qualities for Lena and Woman measured in PSNR are shown in Tables V and VI, respectively. For Lena, PROGRES is slightly better at 0.25 and 0.5 bpp, and SPIHT is slightly better at 0.125 and 1.0 bpp. Meanwhile, for Woman, PROGRES is slightly better at 0.5 and 1.0 bpp, and SPIHT is slightly better at 0.125 and 0.25 bpp. Note that the SPIHT algorithm can stop decoding at an arbitrary point within a bitplane when a target bitrate is met. However, PROGRES always decodes the last available bit on the last bitplane. This property seems to explain the slight difference in coding efficiencies between the two coders.

Figs. 9 and 10 show the reconstructed Lena images and Woman images for four different bit rates by PROGRES and 2D-SPIHT. As stated in the beginning, the two coders decode with very similar quality at the same bit rate.

B. 3D Case

The 3D coding times for 8 bpp gray scale video, Football 8 bpp, $352 \times 240 \times 32$ (SIF format) and Susie 8 bpp $720 \times 480 \times 128$ (ITU 601 format) are shown in Table VII.

The speed gain factor is larger in 3D coding since the amount on the lists being processed in 3D SPIHT

TABLE VII

THE COMPARISON OF CODING TIMES BETWEEN UNCODED 3D-SPIHT AND THE PRESENTED 3D-PROGRES (3D IMAGE SOURCES : FOOTBALL 8 BPP, $352 \times 240 \times 32$ (SIF FORMAT), SUSIE 8 BPP $720 \times 480 \times 128$ (ITU 601 FORMAT))

Bitrate (bpp)	Encoding (in seconds)			Decoding (in seconds)		
	3D-SPIHT	3D-PROGRES	3D-DWT	3D-SPIHT	3D-PROGRES	3D-IDWT
Football						
0.125	0.80	0.46	0.50	0.05	0.01	0.49
0.25	0.89	0.47	0.50	0.10	0.02	0.49
0.5	1.11	0.49	0.50	0.21	0.03	0.49
1.0	1.54	0.51	0.50	0.43	0.04	0.50
Susie						
0.125	13.10	7.27	8.62	0.91	0.13	8.55
0.25	14.87	7.34	8.50	1.82	0.23	8.55
0.5	19.00	7.55	8.56	3.85	0.42	8.55
1.0	26.64	8.03	8.51	7.71	0.80	8.56

TABLE VIII

DECODED QUALITY IN PSNR: 3D PROGRES VS UNCODED 3D SPIHT, CUPRITE_SC1_8B $512 \times 512 \times 224$, 8 BPP

Bit rate	3D PROGRES	3D SPIHT
0.125	26.8938	26.9300
0.25	29.4182	29.4253
0.5	33.0248	32.9183
1.0	37.7560	37.7471

is growing exponentially by dimension. The decoding speed gains are 5 to 10.75 times for Football and 7 to 9.63 times for Susie, as seen in Table VII. The encoding gains are seen as 1.74 to 3.01 times for Football and 1.80 to 3.31 times for Susie.

The decoded qualities for the hyperspectral image, *cuprite_sc1_8b*[2] $512 \times 512 \times 224$, 8 bpp, at four

[2]Upper left 512×512 corner extracted and rewritten to 8 bits per sample from the original 614×512, 16 bits per sample, for each band.

TABLE IX

DECODED QUALITIES AND NUMBER OF BITS FOR 512×512 LENA IMAGE BY PROGRES AT PROGRESSIVE RESOLUTIONS

	Resolution					
	Full (512×512)		Half (256×256)		Quarter (128×128)	
bit rate (bpp)	decoded quality (dB)	decoded size (bits)	decoded quality (dB)	decoded size (bits)	decoded quality (dB)	decoded size (bits)
0.125	30.67	32,768	35.05	31,328	36.4877	22,848
0.25	33.75	65,536	36.02	55,896	36.5062	34,264
0.5	36.89	131,072	36.42	94,016	36.5129	48,792
1.0	39.91	262,144	36.54	149,936	36.5114	65,400

bit rates are also shown in Table VIII. At lower rates, 0.125 and 0.25 bpp, uncoded 3D SPIHT is around 0.01 dB better but at higher rates, 0.5 and 1.0 bpp, 3D PROGRES is around 0.01 dB better. This result tells that both coders give very similar coding efficiency for hyperspectral images.

C. Progressive Decompression

One of the benefits of progressive resolution encoding/decoding is that a reduced number of bits can be decoded to reconstruct a reduced scale. In PROGRES, as shown in Table IX, less number of bits are required to decode at lower resolutions, especially at higher bit rates.

Table X shows that the decoding times of Lena and Woman at 0.5 bpp are increasing for progressively increasing resolutions. In the Lena image, the decoding time increases less than 1.5 times whenever the resolution increases. Meanwhile, the decoding time increases two times for the next higher resolution in the Woman image.

VII. CONCLUSION

The presented coding method, PROGRES (Progressive Resolution Coding), makes extremely fast decoding possible by giving up the quality scalability. Comparing to uncoded 3D SPIHT, it decodes nine times faster, but does not sacrifice the coding efficiency. As shown in the experiments, higher decoding speed gain is expected for larger size images.

TABLE X

DECODING TIME OF PROGRESSIVE RESOLUTIONS, CODED AT 0.5 BPP

(INVERSE WAVELET TRANSFORM TIMES ARE NOT INCLUDED)

Lena (512×512)		Woman (2048×2048)	
Resolution	Decoding time (cycles $\times 10^6$)	Resolution	Decoding time (cycles $\times 10^6$)
16×16	0.4997	64×64	2.1385
32×32	0.5851	128×128	2.9157
64×64	0.8160	256×256	5.5415
128×128	1.5392	512×512	13.2441
256×256	3.0343	1024×1024	35.2448
512×512	4.6403	2048×2048	75.2314

The decision bits during set partitioning of SPIHT are redefined across the bitplanes to give the novel idea of 'hierarchical dynamic range coding', which mainly realizes fast decompression. Using the property of decaying spectral density in wavelet subbands, information of energy decrease across frequency subbands is shared by neighboring coefficients, thereby leading to compact coding of the dynamic ranges.

The given method would be most suitable for applications that need high speed decoding, such as intra-frame decoding in video playback. In addition, due to its inherent simplicity, lower implementation costs both in hardware and software forms are possible.

APPENDIX

THE RELATIONSHIP BETWEEN PROGRES AND SPIHT

There are some similarities and differences between the PROGRES and uncoded SPIHT algorithms. However, remarkably enough, if SPIHT is also given prequantized wavelet coefficients and encodes all bitplanes, every code bit of one algorithm has a corresponding bit in the other algorithm. Consequently, both coders demonstrate very similar coding efficiencies. One may try to understand PROGRES as the coder designed by having a different view than SPIHT, i.e. non-bitplane coding plus dynamic range coding. The differences and similarities between PROGRES and SPIHT are described in the following two sections, with further detailed syntactic compatibility is explained in Section C.

A. Differences from SPIHT

The biggest difference which characterizes the PROGRES from SPIHT algorithm is the selection of 'non-bitplane coding' and 'dynamic range coding'. In PROGRES, each coefficient magnitude is represented by two parts : the number of bits to represent it and the magnitude. The 'number of bits' is understood as the 'dynamic range'. The tree consists of its root value (a wavelet coefficient) and subtrees. Each subtree is recursively defined in the same way as its parent tree. The dynamic range of a tree is represented by the difference d_{base}, from that of its parent tree. Without entropy coding, this difference information is coded as a unary number ending with 0. The end mark '0' can be understood as the 'significance testing bit' in SPIHT. Because, for example, a decrease of dynamic range 3 in PROGRES which is 1110 in unary form can be viewed in SPIHT's context that there are no significant bits for succeeding three bitplanes and the first significant bit will be found at the fourth bitplane from the current bitplane.

Meanwhile, in SPIHT, a significant coefficient is first coded by its position in the significance map, i.e. the bitplane corresponding to current threshold. The position is represented by a sequence of binary decisions of partitions. And then, for the significance bit information in remaining bitplanes at the same position, only the significance for the corresponding threshold is coded without position.

In SPIHT without entropy coding, the magnitude of a significant coefficient can be coded using m bits in m bit-planes. Assume a certain significant coefficient $c_{i,j}$. If the first significant bit of coefficient $c_{i,j}$ is at bit-plane k, where m bit-planes are defined from 0 (LSB) to $m-1$ (MSB), the sorting passes will output $(m-1-k)$ of '0's for bit-plane $m-1$ down to $k+1$ and one of '1' for bit-plane k and the refinement passes will output k of either '0's or '1's for bit-plane $k-1$ through 0 depending on the magnitude of the coefficient. The total number of bits is : $(m-k-1)+1+k=m$. We don't include the set partitioning information to locate the coefficient $c_{i,j}$ since it represents the location information, not the coefficient value itself.

B. Similarities to SPIHT

While there are apparent differences in bitplane management and coefficient magnitude coding schemes between PROGRES and SPIHT, similarities also can be found if we interpret the meanings of each bit used in PROGRES syntax, especially for d_{base} and d_{local}. As stated earlier, we assume both PROGRES and SPIHT encode all the bitplanes of prequantized wavelet coefficients.

First, the similarity between dynamic range coding of PROGRES and sorting pass of SPIHT is stated. The unary value of d_{base} and d_{local} is very similar to the sequence of coefficient significance testings

in SPIHT. That is, each additional '1' of unary value in PROGRES indicates the dynamic range of coefficient is dropped by half, which corresponds to '0' in SPIHT representing 'insignificance' for the given threshold. As an example, if $d_{base} = 1110$ in PROGRES, it means that the coefficient is insignificant for following three less significant bitplanes in SPIHT.

Secondly, the similarity between actual coding of coefficient values in PROGRES and refinement pass of SPIHT is stated as follows. In SPIHT, once the first significant bit of a coefficient appears in certain bitplane, the refinement pass starts for all the next bits of the coefficient. Each bit coded in the refinement pass of SPIHT exactly corresponds to each bit of coefficient in binary form, just starting from the next bit of the first nonzero bit of the coefficient, which is similar to actual coefficient value coding steps in PROGRES.

And the last similarity, which is quite complicated and explained further in Section C , is the zerotree coding scheme in SPIHT and hierarchical dynamic range coding scheme in PROGRES. In SPIHT, two kinds of significance testing exist. One is for coefficients and the other is for sets. The significance testing mentioned in the first similarity above is a significance testing for a coefficient. The '0' in SPIHT indicates 'do not split the set' since the set does not have any significant coefficient in it. And the '1' indicates 'split the set' since the set has at least one significant coefficient in it. Thus, many coefficients grouped in a set are represented together for their significance regarding a current threshold or bitplane, causing savings of many significant bits.

Analogous processing is also done in PROGRES by way of a hierarchical dynamic range coding. Briefly, the decrease in dynamic range between two adjacent resolution levels means that the bits on the bitplanes corresponding to the decreased dynamic range do not need to be coded in the higher resolution levels because they are all zeros implicitly.

C. Syntactic Compatibility between PROGRES and SPIHT

One of the interesting coincidences discovered between two coders is that the two steps of prediction in PROGRES and the two steps of decision in SPIHT have an exact correspondence.

1) The 'dynamic range number' (in PROGRES) of a coefficient (or a tree of coefficients) corresponds to the position (0 for LSB) of the first significant bit (in SPIHT) of the coefficient (or the largest coefficient magnitude in a set of coefficients).
2) The unary coded bits information d_{base} and d_{local} in PROGRES have roles similar to the significance test bits for the type-A set and type-B set in SPIHT.

3) The one significance test bit of a coefficient plus refinement bits in SPIHT correspond to the bit values packed in dynamic range number bits of a coefficient in PROGRES.

In SPIHT, each set in LIS (List of Insignificant Sets), which will be partitioned, is represented by two types: type-A for $D(i,j)$ and type-B for $L(i,j)$ [6]. The definitions of $D(i,j)$, $O(i,j)$, and $L(i,j)$ follow the article [6].

1) $O(i,j)$: Set of offspring of the coefficient (i,j),
2) $D(i,j)$: Set of all descendants of the coefficient (i,j),
3) $L(i,j)$: $D(i,j) - O(i,j)$

The partitioning of a set is represented with binary decisions. If type-A set has any significant coefficient for a given threshold, the coder output '1' meaning there is at least one nonzero bit in $D(i,j)$ of current bitplane. And then, the type-A set is partitioned into subsets and the significance of each child coefficient $(k,l) \in O(i,j)$ is coded [6]. If type-A set outputs '0', it means there is no nonzero bit in $D(i,j)$ of current bitplane and thus the set will not be considered for the current bitplane. In fact, the significance testing of a set in the order of type-A and then type-B can be well understood as the trial for finding a 'wide-sense zerotree', so that a group of many zero bits can be compactly represented. A detailed study on the classification of wide-sense zerotrees such as type-A or type-B of SPIHT has been performed by Cho and Pearlman [21].

A 'set' in SPIHT corresponds to a 'tree' in PROGRES. A set $s_{i,j}$ rooted at (i,j) stays insignificant up to the threshold for which at least one of its coefficients tested as significant. To represent the 'staying insignificant' of the set for several (say, k) bitplanes, SPIHT outputs k '0's. Once the set become significant at certain bitplane, the coder outputs a '1'. At this bitplane, the set is tested for partitioning. If there exists a significant coefficient in any of its subsets, the set is partitioned into subsets that are immediately appended to the end of LIS. From the next bitplane, no bit information is coded for the sets $s_{i,j}$ since the set does not exist anymore. Instead, its subsets or children sets in the LIS, $s_{2i,2j}$, $s_{2i,2j+1}$, $s_{2i+1,2j}$, $s_{2i+1,2j+1}$ if they exist, are tested for the significance at that bitplane.

Assume we have a 4-ary tree (quadtree) with height two, i.e. there are three levels in the tree, where the top is level 0 and the bottom is level 2. And let $c_{i,j}$ be the root coefficient and $c_{m,n}$, $(m,n) \in I(i,j)$ be its four children coefficients at level 1 of the tree ($I(i,j)$ is defined in Section III-B). Then, the magnitudes of all $c_{m,n}$ (i.e. at level 1) is coded with $r_{parents}$ bits in PROGRES. The $r_{parents}$ is represented by the difference information d_{base}. For SPIHT, a type-A set (i.e. $D(i,j)$) is insignificant for d_{base} bitplane passes, and at the next bitplane the set becomes significant and is partitioned so that the bits of its

children coefficients $c_{m,n}$ on the bitplane is coded. Now, a type-A set is converted into a type-B set if there exist any significant coefficient in $L(i,j)$ (i.e. coefficients from level 2 and below). Note that this conversion step is done at the same bitplane where the type-A set become significant.

To code the coefficients at level 2, the PROGRES uses both d_{base} and d_{local} information. In this case, there are four different $d_{local,subtree(s_{m,n})}$, each for predicting the dynamic range number of of $subtree(s_{m,n})$ as described in Subsection III-C. Thus, each coefficient magnitude at level 2 in$subtree(s_{m,n})$ is coded with $r_{children}$ - d_{base} - $d_{local,subtree(s_{m,n})}$ bits in PROGRES. For SPIHT, a type-B set (i.e. $L(i,j)$) is insignificant for d_{base} bitplane passes, and at the next bitplane the set becomes significant and is partitioned into four type-A sets which are appended to the end of LIS. And then, each type-A set $s_{m,n}$ is insignificant for $d_{local,subtree(s_{m,n})}$ bitplane passes, and at the next bitplane the set becomes significant and the same tasks are repeated as above

In the refinement passes of SPIHT, each remaining bit (i.e. below the first significant bit) of every coefficient in LSP (List of Significant Pixels) [6] is coded. One bit for the significance test of a coefficient and refinement bits of the coefficient in SPIHT correspond to the bit values packed in dynamic range number bits of a coefficient in PROGRES. Thus, the number of bits spent to code the magnitude of coefficients is exactly the same.

In this way, we can find the syntactic relations between: 1) the the unary coded bits informationd_{base} and d_{local} in PROGRES and the significance test bits for the type-A set and type-B set in SPIHT, and 2) bit values packed in dynamic range number bits in PROGRES and significance test plus refinement bits in SPIHT.

REFERENCES

[1] B. E. Usevitch, "A tutorial on modern lossy wavelet image compression: Foundations of JPEG 2000,"*IEEE Signal Processing Magazine*, pp. 22–35, Sep 2001.

[2] D. Taubman, "High performance scable image compression with EBCOT,"*IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, July 2000.

[3] M. Marcellin, M. Gormish, A. Bilgin, and M. Boliek, "An overview of JPEG-2000," in*Proc. 2000 IEEE Data Compression Conference*, J. A. Storer and M. Cohn, Eds., 2000, pp. 523–541.

[4] D. Santa-Cruz and T. Ebrahimi, "An analytical study of JPEG 2000 functionalities," in*Proc. of the IEEE International Conference on Image Processing*, vol. 2, 2000, pp. 49–52.

[5] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients,"*IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, 1993.

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

[7] W. A. Pearlman, "Trends of tree-based, set partitioning compression techniques in still and moving image systems," in *Proceedings Picture Coding Symposium 2001 (PCS-2001)*, vol. 5, April 2001, pp. 1–8, invited, keynote paper.

[8] J. Andrew, "A simple and efficient hierarchical image coder," in*IEEE International Conference on Image Processing (ICIP '97)*, 1997.

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

[10] E. Ordentlich, *Private communication*, March 20, 2007.

[11] W. Berghorn, T. Boskamp, M. Lang, and H. O. Peitgen, "Fast variable run-length coding for embedded progressive wavelet-based image compression," *IEEE Transactions on Image Processing*, vol. 10, no. 12, pp. 1781–1790, Dec 2001.

[12] J. Oliver and M. P. Malumbres, "Fast and efficient spatial scalable image compression using wavelet lower trees," in*Proc. 2003 IEEE Data Compression Conference*, Mar 2003, pp. 133–142.

[13] H. Danyali and A. Mertins, "Flexible, highly scalable, object-based wavelet image compression algorithm for network applications," in *IEE Proceedings - Vision, Image, and Signal Processing*, Dec 2004, pp. 498 – 510.

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

[15] S.-T. Hsiang, "Embedded image coding using zeroblocks of subband/wavelet coeffcients and context modeling," in*Proc. 2001 IEEE Data Compression Conference*, Mar 2001, pp. 83–92.

[16] J. E. Fowler, "Embedded wavelet-based image compression: State of the art,"*Information Technology*, vol. 45, no. 5, pp. 256–262, May 2003.

[17] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation,"*IEEE Transactions on Patern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, July 1989.

[18] A. Said and W. A. Pearlman, "Low-complexity waveform coding via alphabet and sample-set partitioning,"*SPIE Visual Communications and Image Processing*, pp. 25–37, Feb 1997.

[19] B.-J. Kim, Z. Xiong, and W. A. Pearlman, "Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-D SPIHT)," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1374–1387, 2000.

[20] J. D. Villasenor, B. Belzer, and J. Liao, "Wavelet filter evaluation for image compression,"*IEEE Transactions on Image Processing*, vol. 4, no. 8, pp. 1053–1060, Aug 1995.

[21] Y. Cho and W. A. Pearlman, "Quantifying the coding power of zerotrees of wavelet coefficients: a degree-k zerotree model," in *2005 IEEE International Conference on Image Processing (ICIP '05)*, Sep 2005, to appear.

Fig. 9. Reconstructed Lena by PROGRES,

(a) 0.125 bpp, 30.67 dB	(b) 0.25 bpp, 33.75 dB
(c) 0.5 bpp, 36.89 dB	(d) 1.0 bpp, 39.91 dB

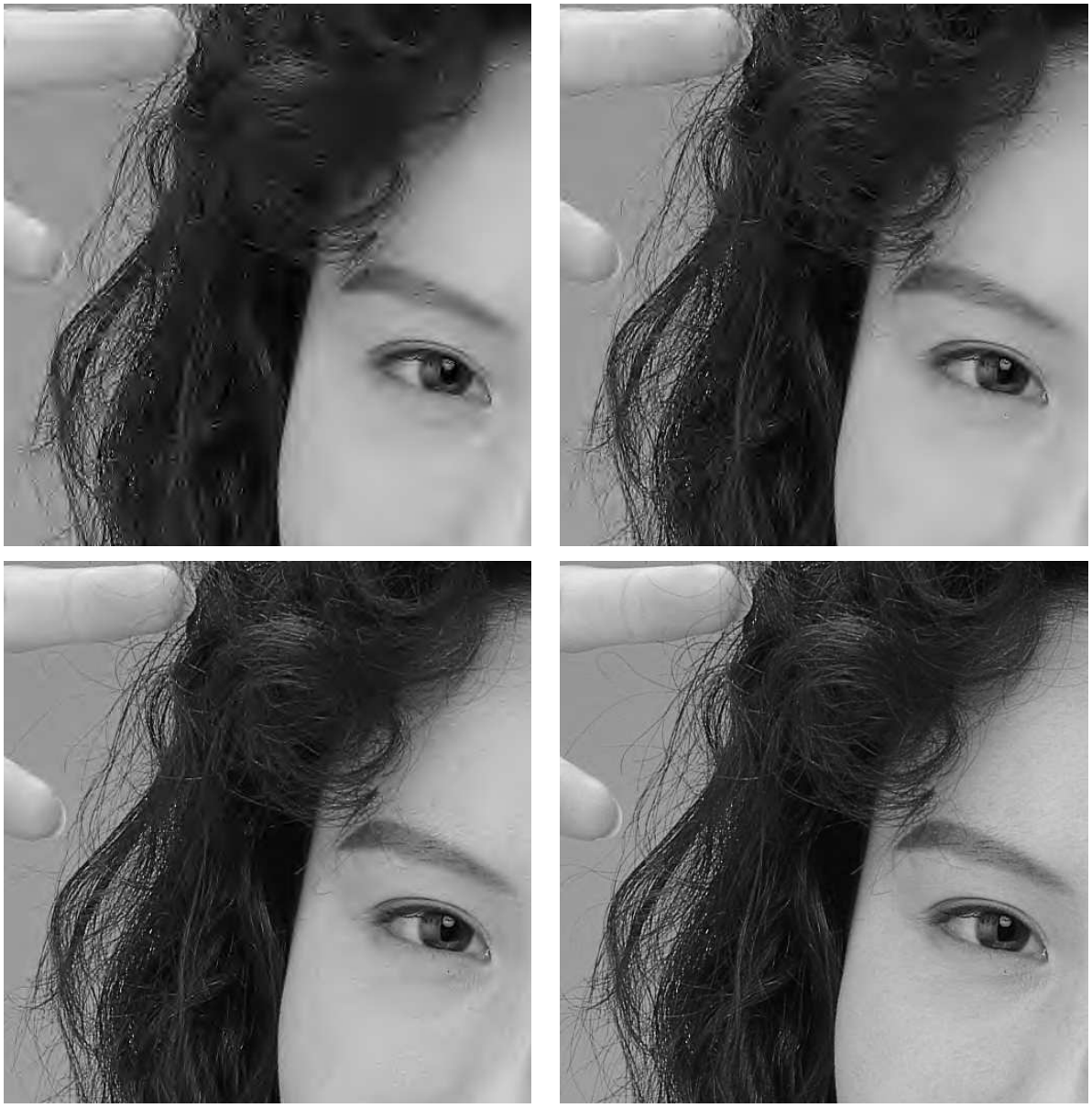

Fig. 10. Reconstructed Woman by PROGRES, 512 × 512 patch at (x,y)=(500,300),

(a) 0.125 bpp, 26.89 dB	(b) 0.25 bpp, 29.42 dB
(c) 0.5 bpp, 33.02 dB	(d) 1.0 bpp, 37.76 dB