

# Fast and constant time random access decoding with $\log_2 n$ block seek time

Yushin Cho<sup>1</sup>, Sungdae Cho<sup>2</sup>, and William A. Pearlman<sup>1</sup>

<sup>1</sup>Rensselaer Polytechnic Institute  
Troy, NY, USA;

<sup>2</sup>Samsung Electronics, Suwon, South Korea

## ABSTRACT

For faster random access of a target image block, a bi-section idea is applied to link image blocks. Conventional methods configure the blocks in linearly linked way, for which the block seek time entirely depends on the location of the block on the compressed bitstream. The block linkage information is configured such that binary search is possible, giving the worst case block seek time of  $\lceil \log_2 n \rceil$ , for  $n$  blocks. Experimental results with 3D-SPIHT on video sequences show that the presented idea gives substantial speed improvement with minimal bit overhead.

**Keywords:** random access decoding, image browsing, interactive imaging, SPIHT

## 1. INTRODUCTION

With the enormously increasing quantity of various image volumes of recent years, most image servers will need to store the volume image data in their compressed form. On the compressed bitstream, the servers also need the ability of search and random access decoding of image data. In addition, for the huge images with more than hundreds of gigabytes, the volume image should be compressed in a ‘tiled’ fashion to avoid ‘trashing’ from the excessive context switching in modern operating systems based on paging systems. For this reason, a tiled version of the 3D-SPIHT (Set Partitioning In Hierarchical Trees)<sup>1</sup> is a good choice for our experiments.

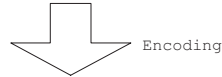
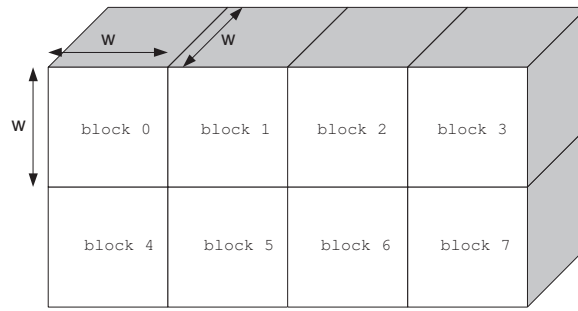
Random access decoding is a scheme for extracting the target information from the bitstream with minimum decoding work. It is usually required in interactive image browsing systems, where users will first browse on coarse resolution images, then probably will look into details of some parts of the images according to their interests. In context-based image retrieval systems, the spatial location of the target image object is often one of the key features to search, where the location is the index of image block on the bitstream.

In the aspect of the image server side, there should seamlessly be many access requests for images or parts of images that need to be fetched from the coded bitstream. Therefore, it is desirable that the access time to a certain part (or block) of the image be well predicted, and best predicted if each access time for the image block is constant. Here, we find the motivation of our research.

### 1.1. Conventional Random Access Decoding : Linear and Slow

Conventional random access decoding methods in JPEG 2000<sup>2,3</sup> simply use a map of indices or links to the blocks (‘code-blocks’ in JPEG 2000 or EBCOT notation). To find the interested code block (or the larger, precinct, tile), the decoder should look up all the indices upto the target block. Consequently, its block seek time depends entirely on the location of a block within a bitstream. It would not be any problem with several megabytes images. However, for huge volume images comprising gigabytes, such as ‘Visible Human Project’,<sup>4</sup> the overall performance of random access decoding in interactive imaging system will depend heavily on block seek time. Rodler<sup>5</sup> presented a representation method of wavelet compressed volume data for fast random access of a voxel. However, it was conducted in the viewpoint of graphics, and therefore the coding efficiency was not competitive.

An image volume with 8 blocks, each block size =  $w \times w \times w$



A compressed bitstream of the image volume with 8 blocks

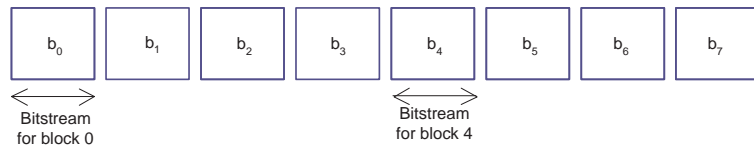


Figure 1. Encoding of an image volume and its bitstream

## 2. RANDOM ACCESS DECODING BASED ON IMAGE BLOCKS : THREE METHODS

An image volume is encoded as  $n$  independently decodable block bitstreams,  $b_i$ ,  $0 \leq i \leq n - 1$ ,  $n = 8$ , as shown in Figure 1. Each bitstream  $b_i$ ,  $0 \leq i \leq n - 1$ , contains the information of a given image block and consists of an positive integer number of bytes, which maybe different for each image block. Three different random access decoding methods are described. Figures 2, 3, and 4 show three different block seek methods: non-random, linear, and bi-sectional, respectively.

### 2.1. Non-random Access Decoding

Full decoding seek with  $n=8$  blocks,  
 average block seek time =  $q \frac{n}{2}$   
 ( $q$ : average time for decoding a block,  $q \gg p$ )

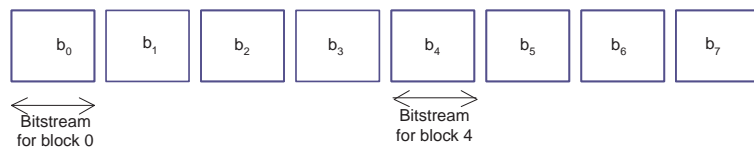
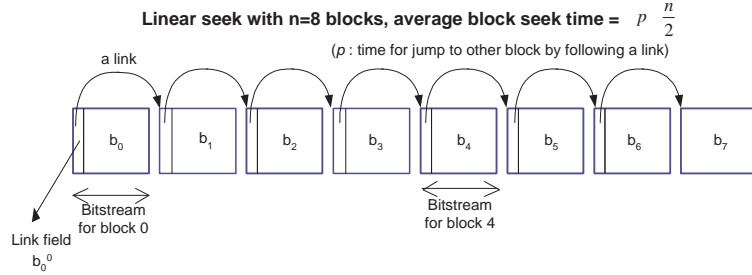


Figure 2. Non-random access block seek on the bitstream of image blocks

In non-random access decoding seek,  $i - 1$  blocks should be decoded to seek the  $i$ -th block,  $b_i$ . Thus, its average block seek time is  $q \times \frac{n}{2}$ , where  $q$  is an average decoding time of a block and  $n$  is the total number of blocks in the bitstream. In Figure 2, if  $b_7$  is requested, 7 blocks (i.e.  $b_0 \sim b_6$ ) should be decoded to reach  $b_7$ .

The block seek time is entirely dependent on where the target block is located on the bitstream. In the worst case, the seek time for the last block (i.e.  $(n - 1)$ -th block,  $b_{n-1}$ ) is  $q \times (n - 1)$ . The complexity of average seek time for non-random method is  $O(n)$  for  $n$  blocks but the constant factor  $q$  is relatively large.

### 2.2. Linear Random Access Decoding



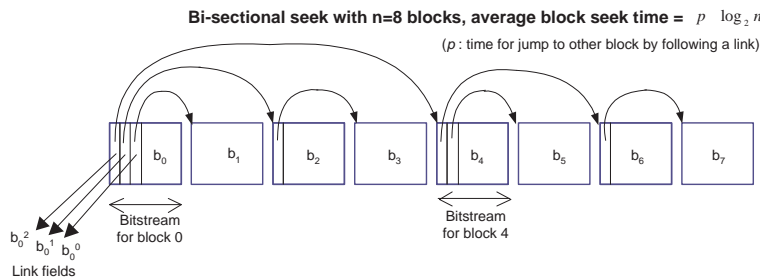
**Figure 3.** Linear block seek on the bitstream of image blocks

Meanwhile, from the beginning of certain block bitstream  $b_i$ , we can directly jump to next block bitstream  $b_{i+1}$  without actual decoding of block  $b_i$  if we know the length of  $b_i$ . This will save the time for random access. In the linear method, the length information of each block bitstream  $b_i$  (except the last block  $b_{n-1}$ ) is stored in the header of block  $b_i$ .

The average block seek time of the linear method is  $p \times \frac{n}{2}$ , where  $p$  is the time for jumping to other block by following a link and it is much smaller than  $q$ . Figure 3 shows  $b_7$  being requested, where 7 links are followed to jump to  $b_7$ . The link from block  $b_i$  to  $b_{i+1}$ ,  $0 \leq i \leq n - 2$ , is represented by the link field,  $b_i^0$  as shown in the Figure 3. The superscript 0 in  $b_i^0$  indicates the range of jump by following a link, which is fixed as one block in the linear method. Assuming the maximum size of bitstream for each image block is  $2^m$  bytes and each block bitstream consists of multiple of bytes, each link can be represented by the link field  $b_i^0$  comprising  $m$  bits. The total number of bits required for link fields is  $(n - 1) \times m$  since there are  $n - 1$  link fields for  $n$  blocks.

As shown in the example, in the worst case, the entire links are followed to reach the last block and the seek time is  $p \times \frac{n}{2}$ , however, which is far less than that of non-random method since  $p \ll q$ . The complexity of average seek time for the linear method is  $O(n)$  for  $n$  blocks and the constant factor  $p$  is very small.

### 2.3. Bi-sectional Random Access Decoding



**Figure 4.** Bi-sectional block seek on the bitstream of image blocks

The suggested bi-sectional random access decoding gives a large improvement over the linear method. Figure 4 shows  $b_7$  being requested, where only 3 links (from  $b_0$  to  $b_4$ , from  $b_4$  to  $b_6$ , and from  $b_6$  to  $b_7$ ) are followed. This is possible since the ‘seek’ to the block is performed similar to ‘binary search’. Three link fields,  $b_2^0$ ,  $b_2^1$ , and  $b_2^2$

are stored in block  $b_0$ , and two link fields,  $b_4^0, b_4^1$  are stored in  $b_4$ . The block  $b_2$  and  $b_6$  has one link field for each. The block  $b_1, b_3, b_5$ , and  $b_7$  does not have link field. The superscript 0 in the link field  $b_i^0$  indicates that the link represented by it has the jump range of  $2^0 = 1$ . For the link field  $b_i^j$  of a block bitstream  $i$ , the superscript  $j$  indicates that the link is used to jump over  $2^j$  bitstream blocks.

If we view the target block index in binary number system, the decision steps for jumping can be understood more easily. Given the  $n$  blocks on bitstream and index range of  $[0..n-1]$ , convert a target index  $j$  into binary number with  $\lceil \log_2 n \rceil$  digits. As an example, block index  $j = 7$  is  $111_2$ . The first 1 indicates we will take the farthest jump (from  $b_0^2$ ). After taking the jump to  $b_4$ , two links,  $b_4^0$  and  $b_4^1$  are available. The second 1 indicates to take the jump from link field  $b_4^1$ , which is the farthest jump available in block  $b_4$ . The last 1 also indicates to take the farthest jump at current position. As another example, if the target index  $j$  were 5, which is  $101_2$  in binary, the second bit 0 indicates not to take jump from block  $b_4$  to block  $b_6$ . Thus, blocks  $b_6 \sim b_7$  are discarded. Now, the last bit 1 indicates to take jump from  $b_4$  to block  $b_5$ , which is the target block we seek.

We decide whether we take the farthest jump starting from the first block  $b_0$ . The farthest jump by following a link from  $b_0^2$  is used to jump over four blocks to reach  $b_4$ . In this way, the first four blocks  $b_0 \sim b_3$  are discarded from the search space by one operation of link-following jump. If this link is not taken, four blocks  $b_4 \sim b_7$  are discarded from the search space. After making the first decision on jumping and its resulting block position, we decide again for next jumping, which will discard the half the subspace remaining. This decision steps are iteratively continued until we arrive at the target block. And the maximum number of decision steps is  $\lceil \log_2 n \rceil$ .

Assuming  $n = 2^m, m \neq 0$ , the bi-section locations, where the links jump to, are given as :  $\{\frac{n}{2}\}$  after the farthest link (i.e. jump over  $\frac{n}{2}$  blocks) is included,  $\{\frac{n}{4}, \frac{2n}{4}, \frac{3n}{4}\}$  after the next farthest links (there are two of them, each of which jump over  $\frac{n}{4}$  blocks) are included,  $\dots$ , and finally,  $\{\sum_{i=1}^{n-1} \frac{i}{n}\}$  after the shortest links are added. Thus, the total number of links is simply  $1 + 2 + 4 + \dots + \frac{2}{n} = n - 1$ . Note that the bi-sectional method requires the same number of links as the linear method uses.

If we assume the maximum size of each block bitstream is  $2^m$ , the total number of bits to represent links or to be required for link fields is :

$$1 \times (m + (\log_2 n - 1)) + 2 \times (m + (\log_2 n - 2)) + 4 \times (m + (\log_2 n - 3)) + \dots + \frac{2}{n} \times m = \sum_{i=0}^{\log_2 n - 1} (2^i \times (m + \log_2 n - 1 - i))$$

for  $n$  blocks. The sequence of number of link field bits,  $\{m + (\log_2 n - 1), m + (\log_2 n - 2), \dots, m + 1, m\}$ , is decreasing by one bit. It is derived from the fact that whenever the jump range in bytes is halved, one bit less is required to represent the range. In other words, if we think from the shortest links, the number of bits to represent the jump link,  $\{m, m + 1, m + 2, \dots, m + (\log_2 n - 1)\}$ , is increasing one bit whenever the jump range is doubled.

Given the block indices are considered as sorted keys in binary search algorithm. The algorithm works on these keys by testing the middle of sorted keys and eliminating the half of the indices interval in which the key cannot exist, and then repeating the procedure iteratively. This search method also can be viewed as 'binary tree'. In particular, for the bi-sectional method presented in this article, a 'complete binary tree' should be constructed. The complexity of binary search algorithm is  $O(\log_2 n)$ . Whenever the algorithm follows a link, half the block indices are eliminated from the search space. This enables the worst case seek time equal to  $p \times \lceil \log_2 n \rceil$ .

## 2.4. Comparison of Three Random Access Decoding Methods

While link information is configured sequentially in conventional methods,<sup>2,3</sup> the suggested method configures link information in a hierarchical way such that a binary search is possible. It runs in a seek time of  $O(\lceil \log_2 n \rceil)$  and guarantees constant seek time even for the worst case, while the conventional linear (next block linking) search method gives average  $O(\frac{n}{2})$  time, where its best case gives  $O(1)$  and worst case gives  $O(n)$ . Best and worst case random access decoding (seek and decode) performance is shown in Table 1.

**Table 1.** Best and worst case random access decoding performance ( $p$  : link following time (sec/link),  $q$  : block decoding time (sec/block),  $p \ll q$ )

Method	Worst case	Average case	Best case
non-random	$q \times (n - 1) + q$	$q \times \frac{n}{2} + q$	$q$
linearly random	$p \times (n - 1) + q$	$p \times \frac{n}{2} + q$	$q$
bi-sectionally random	$p \times \lceil \log_2 n \rceil + q$	$p \times O(\log_2 n) + q$	$q$

The actual average seek time for bi-sectional method notated as  $O(\log_2 n)$  in Table 1 can be derived as :

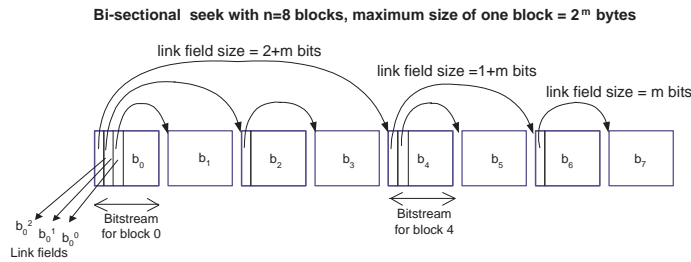
$$\frac{1 \times 1 + 2 \times 2 + 4 \times 3 + \dots + \frac{2}{n} \times (\log_2 n - 1)}{n} = \frac{\sum_{i=0}^{\log_2 n - 1} (2^i \times i)}{n}$$

(jumps/block) for  $n$  blocks, assuming the request for block indices is evenly distributed and  $n = 2^m$ , for  $m \neq 0$ .

### 2.5. Size of a Link Field

In the linear method, the size in bits of a link field is fixed as  $m$  bits, i.e.  $\lceil \log_2 (\max [\text{bitstream size in bytes of a image block}]) \rceil$  bits. I.e. the maximum range of jump to next block is  $2^m$  bytes since the maximum size of a block bitstream is  $2^m$  bytes. Thus, each link field  $b_i^0$ ,  $0 \leq i \leq n - 1$ , is represented by  $m$  bits.

In the bi-sectional method, every link has different size of link field, i.e. some links just jump to next image block (actually, half of  $n - 1$  links are in this use) and some links jump as long as  $\frac{n}{2}$  blocks. Thus, the size of each link field in bi-sectional method is decided corresponding to the jump range of the link. In Figure 5, a link (such as the link from  $b_0^0$ ) jumping to next block is represented by  $m$  bits, a link (such as the link from  $b_0^1$ ) jumping two blocks is represented by  $m + 1$  bits ( $m$  is the maximum bitstream size of a image block in bytes). Thus, the sizes of link fields,  $b_0^0$ ,  $b_0^1$ , and  $b_0^2$  are  $m$ ,  $m+1$ , and  $m+2$  respectively.



**Figure 5.** Size of link fields in bi-sectional method

## 3. PERFORMANCE ANALYSIS BY EXPERIMENTS

Three modes of decoding are experimented : ‘non-random’, ‘linearly random’, and ‘bi-sectionally random’. For encoding, tiled 3D-SPIHT is used with target bit rate around 0.2 bpp and image quality at 29 dB. The block size is set as  $16 \times 16 \times 16$ , which is perhaps smaller than actual in practice, but in order to have many blocks we purposefully used a small block size. In the tiled 3D-SPIHT, each block is byte-aligned and the arithmetic coder is reset for every block, since we need to access a randomly chosen block. The computational platform for experiments is Intel Xeon CPU 2.00GHz with Windows 2000 OS.

**Table 2.** Comparison of random access decoding performances of linear and bi-sectional methods in ‘Susie’ sequence (coded at 29 dB)

Method	Encoding time (secs)	Bitstream size (bytes)	Bit rate (bpp)	Average block seek time (secs)
non-random	67.545	1,252,210	0.2013	38.4183 sec
linearly random	67.828	1,276,509	0.2052	0.0534 sec
bi-sectionally random	67.546	1,300,821	0.2091	0.00058 sec

### 3.1. Comparison of Random Access Decoding Time

A performance comparison is shown in Table 2. Note that average random access decoding time is the average of (block seek time + block decoding time). The block decoding time is 63.2452 ms for all three cases in the table. For the video sequence ‘Susie’ (ITU601, 720x480x144), 12,600 3D-blocks are encoded, where each block size is  $16 \times 16 \times 16$ .

The last column ‘Average block seek time’ in Table 2 shows that the average block seek time for three methods are 38.4 sec, 0.0534 sec, and 0.58 ms (0.00058 sec) for non-random, linearly random, and bi-sectionally random, respectively.

Theoretically, the predicted speed improvement of bi-sectional method over linear one will be  $O(\frac{n}{\log_2 n})$  since the linear method has  $O(n)$  seek time and the bi-sectional one has  $O(\log_2 n)$ . As an example, for the bitstream containing 10,000 image blocks, the suggested method is approximately 100 times faster than the linear method ( $n = 10000$ ,  $\frac{10000}{\log_2 10000} \approx 100$ ).

### 3.2. No Overhead from Encoding the Bi-sectional Links

The computational burden of organizing the hierarchical links in encoder for the suggested fast bi-sectional method is as light as that of the linear method, as illustrated in the second column (encoding time) of Table 2. In encoding time, every link information (i.e. how many bytes to skip to jump to next block) for each block is available only after the image block is encoded.

The difference between linear and bi-sectional method is that the latter accumulates the jump sizes of previous blocks based upon current block index. The task of this size cumulation is managed by a stack, which has maximum depth of  $\lceil \log_2 n \rceil$  at any time. Comparing to the link size forming in the linear method, the additional overhead from pushing and popping over stack is very small. Thus, the overall encoding time for all blocks does not show any noticeable increase.

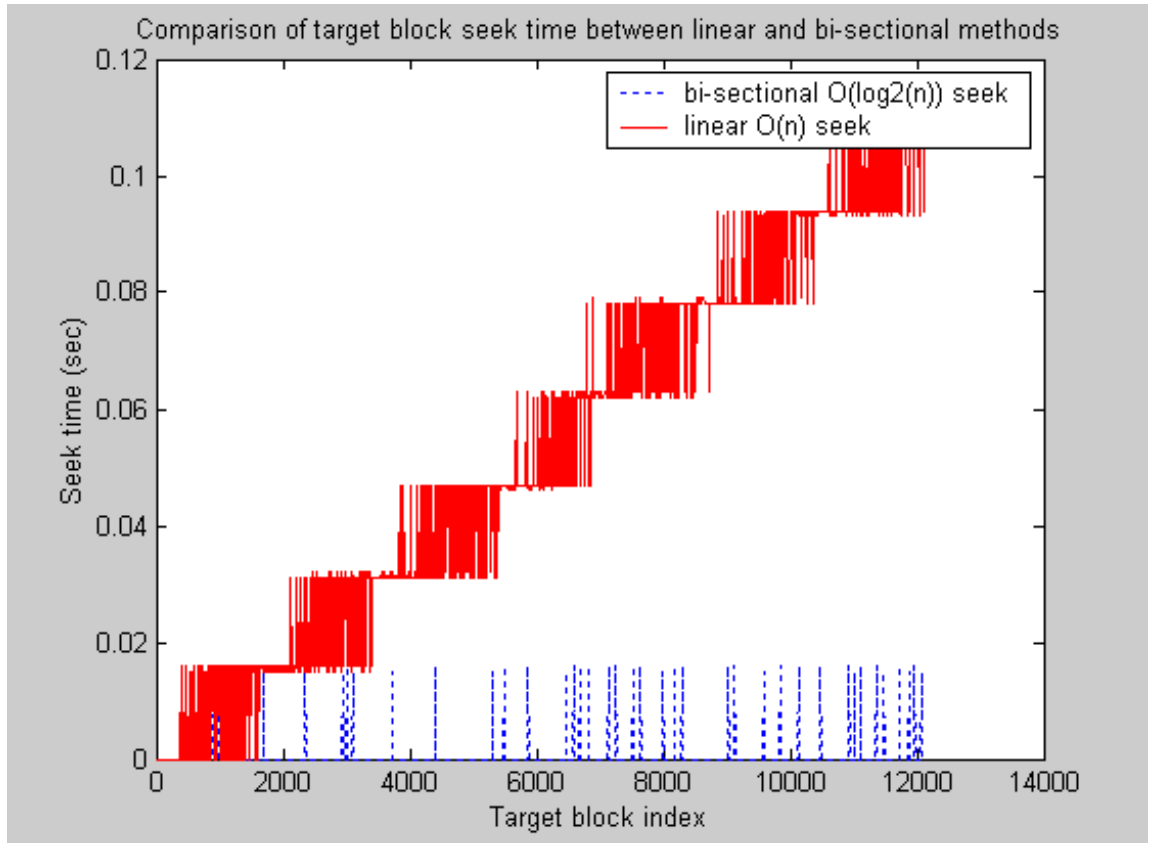
### 3.3. Small Overhead from the Link Information

The generated bitstream sizes are 1.21 MB (1,276,509 bytes) for the linear method and 1.24 MB (1,300,821 bytes) for our bi-sectional method. The difference of two bitstream is 24,312 bytes (0.0039 bpp), less than 2% of the whole bitstream. Thus, the overhead from the link information is relatively small (See Table 2).

Actually, the number of links are the same in both methods but the sizes of some links in our method are larger than those in the conventional linear method. The overhead will decrease for lower bit rate and increase for higher bit rate. In this experiment, the size of each link field is fixed as 2 bytes and 4 bytes for linear and bi-sectional methods respectively. Thus, the maximum allowable size of bitstream in the bi-sectional method is  $2^{32}$  bytes = 4 gigabytes. For longer bitstreams, we may define larger size for link fields. The link fields are not entropy coded.

### 3.4. Experimental Results

As plotted in Figure 6, the performance difference will sharply develop for increasing number of blocks. In the Figure, the seek time curve for non-random access decoding is not plotted because of its low performance. The fluctuations of the two curves are due to the characteristics of memory hierarchy or other tasks in the computer.



**Figure 6.** Comparison of target block seek time between the linear and suggested methods: The horizontal axis shows the target block index that user want to decode, and the vertical axis shows the spent time in seconds to seek that block, just before the decoding. The  $n$  is the total number of blocks in the bitstream, which is 12,600 in this plot.

The overall results from experiments show that the suggested method is superior to conventional linear seek method. The advantage of the suggested bi-sectional seek method is that substantial improvements on seek speed can be obtained with very small bit overhead and without any extra encoding time over the linear method. Executable decoders for random access decoding and test bitstreams are downloadable from [http://www.cipr.rpi.edu/~choy/quick\\_random\\_access/](http://www.cipr.rpi.edu/~choy/quick_random_access/).

#### 4. JUSTIFYING WORST CASE PERFORMANCE

Assuming the number of blocks  $n = 2^m$ ,  $m \neq 0$ , the maximum number of steps of link following will be required for the block  $b_{\frac{n}{2} + \frac{n}{4} + \dots + 2 + 1}$ , which is the last block  $b_{n-1}$  on the bitstream. Thus, the worst case seek time is  $p$  (sec/link)  $\times \log_2 n$  links, where  $p$  is the time for one jump by following a link. Actual block decoding time will add an average block decoding time  $q$  (sec/block) to this quantity. If  $n \neq 2^m$ ,  $m \neq 0$ , the worst case seek time will be not more than  $p$  (sec/link)  $\times \lceil \log_2 n \rceil$  steps.

#### 5. CONCLUSION

An order,  $O(\frac{n}{\log_2 n})$ , faster method for random access decoding out of  $n$  image blocks is presented. The dramatic speed improvement over the conventional linear method is straightforward. In order to seek a target block designated by a block index, instead of following average  $\frac{n}{2}$  links in linear method, the suggested method follows at most  $\lceil \log_2 n \rceil$  links. With small sacrifice of bit overhead from link configuration on the bitstream, the seek

operation to a target block is always finished within  $\lceil \log_2 n \rceil$  steps of link following by the presented bi-sectional seek method. There is no significant increase in the amount of encoding time. This is the fastest random access decoding method known to date.

## REFERENCES

1. B.-J. Kim, Z. Xiong, and W. Pearlman, "Low bit-rate scalable video coding with 3-d set partitioning in hierarchical trees (3-d spht)," *IEEE Trans. on Circuits and Systems for Video Technology*, pp. 1374–1387, 2000.
2. M. Marcellin, M. Gormish, A. Bilgin, and M. Boliek, "An overview of jpeg-2000," *Proc. Data Compression Conference*, pp. 523–541, 2000.
3. D. Santa-Cruz and T. Ebrahimi, "An analytical study of jpeg 2000 functionalities," *Proc. of the IEEE International Conference on Image Processing* **2**, pp. 49–52, 2000.
4. NLM, *The Visible Human Project*, United States National Library of Medicine, National Institutes of Health, [http://www.nlm.nih.gov/research/visible/visible\\_human.html](http://www.nlm.nih.gov/research/visible/visible_human.html).
5. F. F. Rodler and R. Pagh, *Fast Random Access to Wavelet Compressed Volumetric Data Using Hashing*, BRICS Report Series RS-01-34, Dept. of Computer Science, Univ. of Aarhus, Denmark, 2001.