REAL TIME SOFTWARE IMPLEMENTATION OF SCALABLE VIDEO CODEC

W. Tan, E. Chang, and A. Zakhor

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720 USA
E-mail: {dtan,changed,avz}@eecs.Berkeley.EDU

ABSTRACT

Scalable video compression is becoming increasingly more important in diverse, heterogeneous networks of today. In our previous work [2], we developed a scalable codec capable of generating bit rates from tens of kilo bits per second to several mega bits per second with fine granularity of available bit rates. This codec is based on 3-D subband coding and multi-rate quantization of subband coefficients, followed by arithmetic coding. In this paper, we will replace the arithmetic coding portion of the codec in [2] with block coding, and compare encode/decode speed of this new coder with MPEG. Unlike MPEG, this codec requires symmetric computational power at the decoder and encoder and as such is useful in software only, real time, interactive video applications. We have found the encoding speed of the new encoder to be one order of magnitude faster than MPEG-1, without significant loss in compression efficiency.

1. INTRODUCTION

With ever more video information stored in digital format, real-time video encoding and decoding has become an indispensable capability for workstations. The simplest application would be playing back video archives from either a CD-ROM or over other storage devices. Another application would be video conferencing. Currently, most real-time decoding mechanisms resort to expensive special purpose hardwares that become outdated quickly. A software implementation is clearly superior not only in being cheaper, but also in being easily portable to virtually any machine with performance scaling automatically with processing power.

Real-time MPEG decoding is already happening [3]. However, MPEG lacks the multi-rate, multi-resolution capability as often desired for transmission over heterogeneous networks. Recently a scalable video compression algorithm has been proposed [2] that not only operates at a wide range of bit rates, from tens of kilo bits to several megabits per second, but also provides a fine granularity of available bit rates. In addition, these properties are achieved without loss of compression efficiency as compared to standard algorithms such as MPEG. However unlike MPEG where the encoding is considerably more compute intensive as compared to decoding, the scalable algorithm in [2] requires symmetric computation power in encoding and decoding. This codec is based on 3-D subband coding and multi-rate quantization of subband coefficients, followed by arithmetic coding. This symmetric encode/decode capability is a direct consequence of the fact that unlike MPEG, there is no motion vector estimation in the 3-D subband codec.

In profiling the encode/decode times of the codec in [2], we find arithmetic coding to be a major bottle neck. For instance decoding one SIF size frame at 500 kilo bits per second using a 170 Mhz Ultra Sparc requires 91 msec, of which 51 mili seconds is devoted to arithmetic decoding. In this paper, we improve speed limitations of arithmetic coding portion of the codec in [2] by replacing it with block decoding. This new codec still enjoys symmetric encode/decode properties, but more importantly, can achieve encode speeds of up to one order of magnitude faster than MPEG-1, without significant loss in compression efficiency. The outline of the paper is as follows. Section 2 describes the way block coding is applied to layered quantization, section 3 describes implementation details, and section 4 describes speed and efficiency comparisons with MPEG.

2. ZERO CODING BASED ON HIERARCHICAL BLOCK CODING

One of the most compute intensive parts of the codec in [2] is arithmetic coding of multi-rate quantized 3-D subband coefficients. In this section, we will propose block coding as an alternative to the arithmetic coding of subband coefficients.

In multi-rate quantizing subband coefficients, we choose a dead zone quantizer in which the width of the deadzone is twice as large as the width of each

This work was supported by NSF grant MIP-9057466, ONR grant N00014-92-J-1732, AFOSR contract F49620-93-1-0370, California State Program MICRO, Philips, SUN Microsystems, LG electronics and Tektronix.

0-7803-3258-X/96/$5.00 © 1996 IEEE

quantization bin, for all quantization layers. In successive quantization of subband coefficients, each non-deadzone quantization bin is divided into two equal size bins. Since a large number of coefficients in 3-D subband coding are close to zero, and hence fall into the dead zone, coding the "significance map" for each quantization layer is an important issue. By significance map of a quantization layer, we mean a binary map showing the location of coefficients in the deadzone, versus those outside the deadzone. An example of significance maps for quantization layers zero and one are shown in Figure 1. As seen the significant coefficients in layer i are a subset of those in layer $i+1$. As we will see later, we can exploit this for efficient coding.

We will now show how hierarchical block coding techniques can be used to code significant maps. The basic hierarchical block coding technique we use was presented in an early work by Kunt [1] for two-level images. Kunt's method begins by partitioning an image into 16×16 blocks. If the block contains all zeros, the block is coded as a "0", and the algorithm proceeds to the next block. Otherwise, the block codeword begins with a "1", and the block is subdivided into four 8×8 blocks, each of which are coded the same way. In this manner, the coding proceeds in a recursive manner until 1×1 blocks We show an example of coding the first two layers of an 8×8 block in Figure 1. As seen, the initial layer, layer 0, is coded using Kunt's original method. The "Bits:" indicate the coded bitstream, and the "Size:" indicates the size of the block corresponding to the above bit.

To code the next layer, we use the information in the previous layer to avoid coding redundant bits. Specifically, any bits that are marked "1" in the previous layer are also assumed to be "1" in the following layer. For example, consider layer 1 in Figure 1. We assume the decoder has both bitstreams for layers 0 and 1, and layer 0 has been successfully decoded. To decode layer 1, the decoder cycles through the layer 0 bitstream again, filling in needed information as follows. The first task in decoding layer 1 is to decide whether or not the entire 8×8 block has any significant bits. Since layer 0 has significant bits, and layer 1 is a superset of layer 0, layer 1 must also have some significant bits. Therefore, the decoder assumes a "1" for the size 8 bit and does not require additional information. Since the decoder does not require additional information, the encoder will not send any; this is indicated by the "-" in the size 8 bit for layer 1, indicating that no bits are sent. The same process occurs for the first 4×4 block: the corresponding 4×4 block in layer 0 is significant, so nothing is coded for layer 1. The first 2×2 block, however, is empty in layer 0, and so the decoder does not know a priori whether the block contains any pixels in layer 1. Thus one bit must be sent to encode that information. As seen, the block is empty in layer 1 also, and

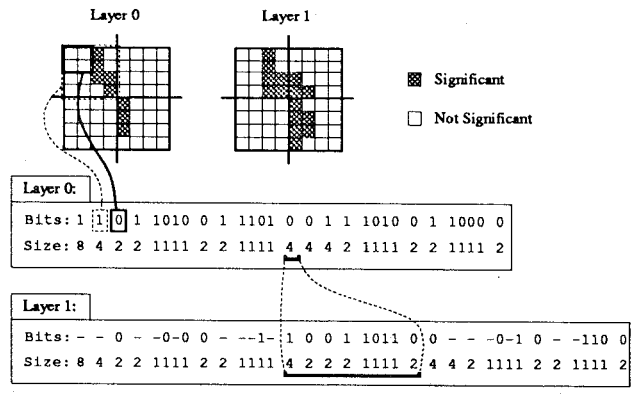

Figure 1: Example of two layers of block coding

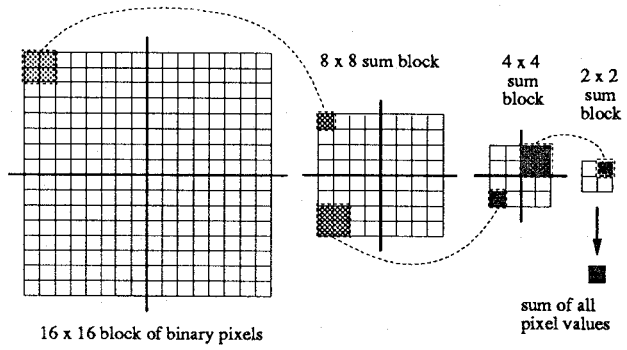

Figure 2: Straightforward implementation pyramid

so a 0 is coded for the size 2 bit in layer 1. Note that this is the first coded bit in the layer 1 bitstream, as it is the first piece of information that is not completely known from layer 0.

3. IMPLEMENTATION ISSUES

In practice, we only use block coding to code significant maps in quantization layers in which it is more efficient than simple binary coding. In this section, we will describe implementation issues related to block coding and subband filtering.

3.1. Simple Implementation of Block Coding

In our straightforward implementation of hierarchical block coding in the previous section, we store each 16×16 block as a 2-D 16×16 array of 8-bit characters. We then construct the 5-level pyramid data structure shown in Figure 2. As seen, each element in the $N \times N$ sum block of layer i is the sum of four pixels from the $2N \times 2N$ sum block of layer $i-1$.

We then use this pyramid structure of sum blocks to compute the coded bitstream as follows. The first coded bit indicates whether there are any significant

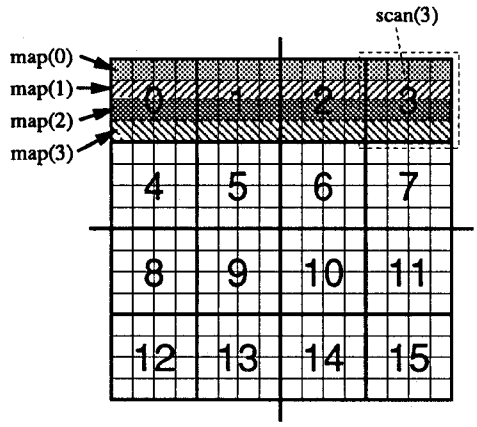

Figure 3: Optimized memory rescan

pixels in the entire original 16 × 16 block, so we call this bit the "size 16" bit. Since the final sum in the pyramid is equal to the sum of the pixels in the original 16 × 16 block, we can code the size 16 bit as a "0" if the final sum is zero or "1" if the final sum is positive. If the first bit is a "1," we next code each of the four 8 × 8 sub-blocks in each quadrant of the 16 × 16 block recursively. Coding proceeds in this manner until the entire 16 × 16 block is coded. Using this straightforward procedure, only one comparison is used to generate each coded bit. However, generating the pyramid is relatively time-consuming; to create an $N \times N$ sum block, we need to perform $4N^2$ memory reads to access the elements to be added, $3N^2$ additions, and N^2 memory writes to store the sum block.

3.2. Optimization of Hierarchical Block Coding

To optimize our implementation, we minimize the number of memory operations by storing each 16 × 16 block as a 1-D array of sixteen 16-bit short integers. This reduces the number of memory accesses required to read the entire array from 256 to 16. For each 16 × 16 block, we use bit masking to rescan each short integer to contain 4 rows of 4 pixels each. We call the original array "map" and the rescanned array "scan". The rescan operation is shown in Figure 3; for example, the fourth element of the scan array is denoted "scan(3)" and composed of the last four bits of each of the following map elements: map(0), map(1), map(2), and map(3). Thus we can access the original 16 × 16 block in 4 × 4 sub-blocks, each of which corresponds to one element in the scan array.

Using the scan array, our optimized coding algorithm proceeds as follows. To code the size 16 bit, we must determine whether the original 16 × 16 block contains any significant pixels. This is equivalent to checking whether any of the 16 elements in the scan array are nonzero. One way of doing this would be to access all 16 scan elements, apply 15 bitwise OR operations, and check the result for a nonzero value. Alternatively, we can use a trick to halve the number of memory accesses by assigning a 32-bit integer pointer to the beginning of the scan array. Since a 32-bit integer can contain the data of two 16-bit short integers, we only need to check 8 consecutive integers using 7 bitwise OR's to test if all 16 elements of the scan array are zero. We code the size 16 bit as a "0" if the value of the resulting OR's is zero, "1" otherwise. Similar approaches can be used for size 8 and size 4 bits.

The breakdown of decode times for one video frame at 500 kilo bits per second on a 170 Mhz Ultra Sparc workstation indicates that of the 50 mili-seconds total decoding time, only 20 miliseconds is devoted to block decoding.

3.3. Fast Subband Filtering

Three dimensional subband analysis and synthesis are performed using separable applications of one dimensional digital filters. In general, using a N tap filter requires N multiplications, $N-1$ additions, N memory loads and 1 memory store for every point in the input data. Linear phase filters are typically used which reduces the number of multiplications to $\lfloor N/2 \rfloor$. In today's workstations it is not uncommon to have a large number of registers where the multiplications and additions can be very efficiently executed, making memory accesses a major cost of subband filtering. Since $N-2$ of the N input samples needed to compute the output at time k are identical to the input samples needed to compute the $(k+1)st$ output sample, we can exploit the register structure to only load two input sample from memory per output points. As it turns out the same two new input samples can be used for both the low and high frequency subbands. Hence for every output sample of high and low frequency subbands, we require only two additional memory loads. Similar approach can be used to reduce the number of memory loads for subband synthesis.

4. PERFORMANCE COMPARISON WITH MPEG-1

In this section, we compare the performances of the scalable codecs to that of MPEG-1. We use the Berkeley Plateau Multimedia Group MPEG-1 codec [3], v. 1.5b, with search range of ±7 pixels and a GOP pattern of IBPB, except for $md-mpeg$ at 3Mb/s in Table 2 which uses IP. The number of spatial decomposition is 4 for the luminance component and 3 for each of the chroma components of the scalable codec. The rate control was chosen in such a way as to achieve constant bit rate over each GOP in MPEG and equivalent

R (kb/s)	64	256	500	1000	1500	3000
$rd-t2-bc_e$	24.0	24.0	20.0	15.2	12.5	9.9
$rd-t2-bc_d$	24.0	21.4	18.5	14.5	12.8	10.6
$rd-t2-ac_d$	24.0	18.7	11.0	7.0	5.4	3.9
$mpeg_d$	24.0	24.0	24.0	24.0	24.0	24.0
$mpeg.exh_e$	0.4	0.4	0.4	0.4	0.4	0.4
$mpeg.log_e$	1.6	1.6	1.6	1.6	1.6	1.6
$rd-t1-bc_d$	24.0	24.0	19.9	15.6	13.8	10.8
$pp-t1-bc_d$	24.0	20.6	16.7	13.7	11.9	9.2
$fb-t1-bc_d$	24.0	21.3	17.0	13.6	11.8	9.6
$md-t1-bc_d$	24.0	21.3	18.0	14.4	12.4	9.5

Table 1: Encoding and decoding speed comparison.

Rates Mb/s	0.5	1 (t2)	1 (t1)	1.5	3.0
$rd-mpeg$	30.9	34.1		35.9	38.9
$rd-bc$	31.7	35.2	35.2	37.0	39.3
$pp-mpeg$	25.9	28.5		30.3	33.9
$pp-bc$	25.1	28.4	26.8	30.1	33.0
$fb-mpeg$	30.2	33.1		34.9	38.0
$fb-bc$	31.0	34.1	34.3	35.9	38.0
$md-mpeg$	36.0	38.7		40.7	42.9
$md-bc$	37.0	40.4	38.9	42.1	45.0

Table 2: PSNR comparison.

GOP for the subband codecs.

Table 1 shows the speed performance of MPEG and the 3-D subband codec for four different sequences at six different rates. *rd* stands for the sequence "Raiders of the lost ark", *pp* for "Ping Pong", *fb* for "football" and *md* for "Mother Daughter". Except for Raiders that is 320×240, all the other sequences are 352×240. $t-1$ and $t-2$ denote one and two layers of temporal decompositions respectively. *ac* stands for scalable coding with arithmetic coding and *bc* stands for scalable coding with block coding. $mpeg.exh_e$ and $mpeg.log_e$ stand for MPEG encoding with exhaustive and logarithmic search respectively. The speed tests were done on a 170 Mhz ultra sparc workstation. The decoding speeds include disk access from a local disk, dithering and display, while the encoding speeds exclude disk access. This is because in most real time encoding applications, raw video will be read from a video camera and not disk. Finally, multi-threading techniques were used in order to overlap blocking operations of compressed video input and video frames output with actual compression, providing better speed performance.

As seen, depending on speed, the subband codec with block encoding is 20 to 60 times faster than exhaustive search MPEG encoding. Even though using logarithmic instead of exhaustive search speeds up MPEG encoding by a factor of 4, it is still considerably slower than scalable codec with block coding. The speed of both scalable codecs, based on arithmetic and block coding are for the most part symmetric with respect to encoding and decoding. In practice, the decoding is slightly slower than encoding because it has to deal with clipping, YUV to RGB conversion, dithering and display. As seen, the block coding approach is up to twice as fast as arithmetic coding approach. Decreasing the number of temporal decompositions from 2 to 1, speeds up the encoding/decoding of the block coder. Finally, there is some variability across sequences as far as speed is concerned. This can be attributed to varying amount of motion in the three sequences, and the fact that the Raiders sequence has 10 % fewer pixels than the other sequences.

Table 2 shows the luminance PSNR performance of the 3-D subband codec using block coding and MPEG-1 for four video sequences at rates 0.5, 1, 1.5, and 3 Mb/s. The scalable codec uses two layers of temporal decomposition unless otherwise stated. It is important to emphasize that for the scalable 3-D subband codec one bit stream at 3 Mb/s is generated once and its subsets are extracted to obtain other bit streams at other bit rates. On the other hand, for MPEG, a whole different bit stream is generated at the encoder for each bit rate. As seen, except for Ping Pong, the scalable codec performs as good or better than MPEG codec for the other three sequences. Decreasing the number of temporal decompositions from two to one sometimes adversely affects the SNR, and improves encode/decode speed.

Finally, note that it is possible to improve the PSNR of the 3-D scalable codec described here, by applying pan compensation techniques described in [2]. However, this is at the expense of increased computational complexity at the encoder.

To summarize, the proposed scalable codec outperforms MPEG in encoding speed by an order of magnitude without significant loss in PSNR. The scalable codec with block coding also achieves reasonable decoding speeds, making it a viable choice for real time, software only, interactive video compression applications.

5. REFERENCES

[1] M. Kunt, "Block Coding of Graphics: A Tutorial Review," *Proceedings of the IEEE*, Vol. 68, No. 7, July 1980

[2] D. Taubman and A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Transactions on Image Processing*, Sept. 1994, vol. 3, no. 5, pp. 572-588

[3] K. Patel, B. Smith, L. Rowe, "Performance of a Software MPEG Video Decoder," *Proceedings of the ACM Multimedia '93*