A REAL TIME SOFTWARE-ONLY H.261 CODEC

Katherine Wang, James Normile, Hsi-Jung Wu,
Dulce Ponceleón, Ken Chu, and Kah-Kay Sung

Advanced Technology Group
Apple Computer, One Infinite Loop, Cupertino, CA 95014

ABSTRACT

Video and audio conferencing over networks is becoming increasingly popular due to the availability of video and audio I/O as standard equipment on many computer systems. So far, many algorithms have concentrated on playback only capability [1]. This generally results in unacceptable real-time performance with respect to latency and encoder complexity. We describe a software-only system that allows full duplex video communication. For our analysis and implementation we chose a DCT based method that uses motion estimation and is modelled on the CCITT H.261 standard.

We begin with a brief discussion of the algorithm and follow with an analysis of the computational requirements for each major block. The results presented show the effect of computational simplifications on signal to noise ratio and image quality. In our conclusion, we examine the processing needs for full resolution coding and project when this will become available.

1. INTRODUCTION

The conventional approach to teleconferencing has been to use dedicated hardware for compression and decompression. This viewpoint assumes that the computational requirements are beyond the capabilities of general purpose processors. Some of our earlier work [2] showed how a small number of specialized signal processors could be used to achieve real-time compression and decompression. Since then there has been a four to eight fold increase in the performance of general processors. Our goal is to provide real-time simultaneous encode and decode on a single general purpose processor.

We chose to implement the CCITT H.261 [3] video codec because of its widespread acceptance as a video conferencing standard and its ability to deliver good quality video at low bit rates. Undoubtedly, higher quality compression techniques will be developed in the future; however, in our opinion the H.261 standard offers the best overall performance at present. The target characteristics of our codec are 10 frames/sec of QCIF 176x144 sized frames simultaneously encoded and decoded at bit rates from 64 kbits/s to 384 kbits/s. The size and frame rate are near the lower threshold for acceptable visual performance.

The H.261 encoder consists of a forward path comprised of motion compensated error computation, followed by DCT, scalar quantization and lossless coding and a feedback path using inverse quantization followed by inverse DCT and frame integration. Table 1.1 shows a breakdown of the main computational blocks. Clearly motion estimation dominates the numbers. As a first approximation, we removed motion estimation and allowed only intra or inter block coding. This required 3-5 times the bandwidth to achieve the same SNR level [4]. Even with the increased bandwidth we felt that the image quality was unacceptably poor. In the final implementation, we use a sparse motion estimation technique which provides much of the benefit of exhaustive search at a small fraction of the computational expense.

1.1. Compute Requirements

The computational requirements for a standard H.261 implementation is prohibitively high for today's class of personal computers. Table **1.1** shows a breakdown of the operations per second required for the major functional blocks of the algorithm assuming QCIF (176x144, 10) frames/s, and exhaustive search motion estimation within the +/- 15 pixel search range. The inverse DCT, inverse quantization and inverse color transformation operations are assumed to have the same computational requirements as their forward counterparts. No estimate is given for the Huffman encoding, which

Table 1.1 Breakdown of Compute Requirements

		Mult/frame	Adds/frame
DCT		123552	275616
Quantization		38016	38016
Inverse Quantization		38016	76032
Motion Estimation			22809600
Motion Compensation			38016
Huffman Decoding		89100	
RGB to YUV Trans		126720	101376
Decoder		288288	580140
Encoder	(no Mot. Est)	576576	906048
	(with Mot Est)	576576	23715648

Table **1.1**: Estimates for codec computational requirements. From these numbers it appears to need about 15 million operations per second for full duplex encode and decode. Practically these operations are rarely if ever completed in one cycle. I/O operations are also ignored. Our experience is that it requires 2-4 times the expected compute to perform the operations. The decoder computation is about half that of the encoder (without motion estimation).

0-7803-2431-5/95 $4.00 © 1995 IEEE

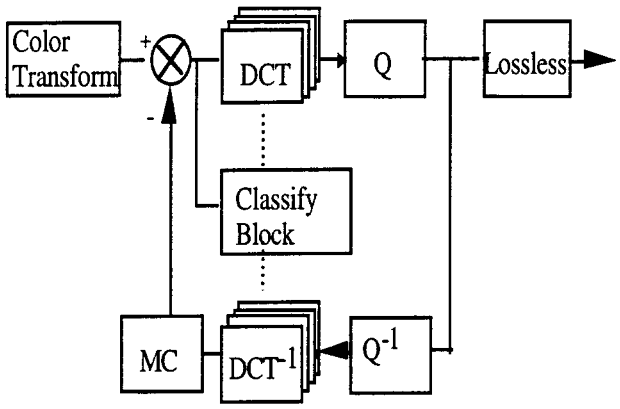


Figure 2.2(a): Encoder Structure.

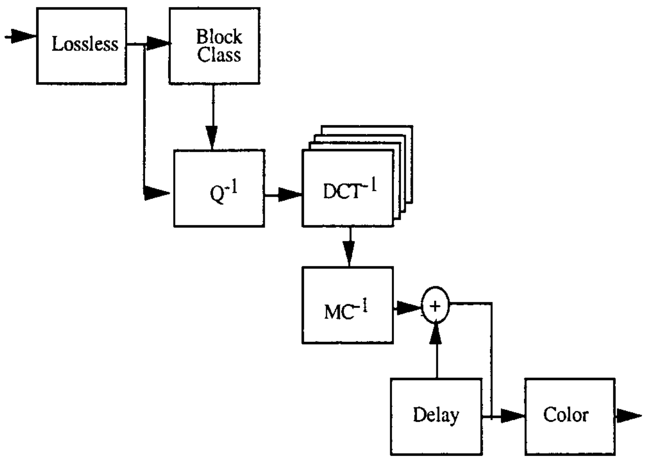


Figure 2.2(b): Decoder Structure.

is computationally simpler than Huffman decoding.

2. CODEC STRUCTURE AND ENHANCEMENTS

2.1 Overview

We generate bitstreams which are compatible with the H.261 standard. However, we have adapted the algorithm to be suitable for software-only implementation. Figure 2.1 and 2.2 show the encoder and decoder structure. A block classifier chooses among a number of different DCTs and inverse DCTs. We have combined the DCT and quantization stages to minimize the number of memory accesses. Analogous optimizations are performed in the inverse DCT and the inverse quantization stages. In the following sections we show how the complexity of the motion estimation, the transform, and the lossless coding stages can be reduced. Performance measurements for individual blocks are presented in [5].

2.2 Motion Estimation

If a full search, block matching algorithm is used, motion estimation is the single most expensive stage. In our current encoder, we use a low-complexity motion estimator which may yield suboptimal vectors but takes less than 10% of the encode time.

We have made a number of simplifications to the conventional block matching scheme. (1) In our scheme, motion estimation is performed on a sparse grid of macroblocks. Vectors for interstitial macroblocks are interpolated from the surrounding vectors of macroblocks on the grid. (2) For each macroblock on the grid, we use a hierarchical block-matching scheme. That is, we sample the search area coarsely and match on those points. From the best match, we define a smaller search area and repeat the process more finely. The process is recursively performed until the search area degenerates to one point. (3) Furthermore, the block matching is performed only on half of the pixels in the block. The candidate pixels are chosen in a checkerboard pattern. (4) Finally, the search is terminated when the motion compensated error falls below a threshold.

2.3 Adaptive transform

The Discrete Cosine Transform (DCT) has been extensively used for image compression because the transform compacts the energy effectively into lower frequency coefficients for typical images. We exploit this fact in our block categorization mechanism which is described later.

The DCT has received much attention over the last ten years and various optimizations for its efficient calculation have been introduced. We implemented a variety of methods before settling on the Chan-Ho approach [6]. In an initial test of functionality we used the Chen algorithm which is separable, ie. it computes a one dimentional DCT for every row and column of a block.

The second version of the codec used Arai et. al.'s algorithm. In this method, the DCT coefficients are not computed explicitly and require only five multiplications for an 8-point DCT. A further scaling step is needed for the explicit calculation of the coefficients. Arai et. al.'s approach is useful when the scaling step can be combined with the quantization, this is the case for JPEG. For H.261 we were unable to find an efficient way to merge the scaling and quantization. Our approach of using large table lookups proved inefficient in its use of the RISC architecture.

The Chan-Ho algorithm, a non-separable approach, extends a 1-D technique to 2-D by using a vector-radix decomposition. The main advantage of this method is its computational efficiency. It requires 25% less multiplications than the separable approach of Chen et. al.

We reduce the computation when the input block or the transformed coefficients exhibits a simple form. The codec distinguishes among fours types of 8x8 blocks: those with all zeros (Zero Blocks), blocks with only the DC coefficient (DC Blocks), 8x8 blocks which have nonzero transform coefficients only in the lowest frequency quadrant, a 4x4 of the 8x8 (4x4 Blocks), and regular blocks (8x8 Blocks). In the encoder we detect whether a regular 8x8 block can be approximated by a simpler type. This detection is performed by computing statistics on the input 8x8 block. In particular, we define block energy measurements to capture the magnitude of the energy in the block and whether it is principally high or low frequency energy. An adaptive control mechanism set thresholds to control the percentages of various block types used in the encoding process. At the decoder, special block types are detected after the entropy decoding stage and simplified inverse quantization and transformation are performed according to the block type detected. We can trade off speed for quality by adjusting the fraction of each block type allowed. Clearly in the case of a skipped block, no further computations are required for encoding or decoding. Similar simplifications occur for DC only block types.

Tables 2.2a and 2.2b show the different block types resulting naturally from DCT and quantization. All blocks are encoded as standard 8x8 Blocks, the decoder detects the resulting block type and uses the appropriate IDCT and inverse quantizer. Decoder block type detection is lossless. Note that for lower bitrates the decoder already detects many transformed blocks to be non-8x8, hence forcing non-8x8 block types in the encoder does not result in a significant speedup.

2.4 Entropy Coding

Huffman coding is performed on block headers and on the runlength coded transform coefficients. The encoding process is fixed-to-variable length and is implemented as table look ups. The decoding process is variable to fixed length and is slightly more complicated. We used multilevel look up tables [8] in the Huffman decode. The idea is to simultaneously decode multiple bits. To start the process, a block of N bits is grabbed from the bitstream. If one token can be decoded from the block, then the token is decoded, and the unused bits are returned to the bitstream. If more bits are needed to decode a token, a second block of M bits is grabbed from the bitstream. As many blocks as necessary are grabbed from the bitstream until a token is decoded. Unused bits are replaced in the bitstream.

We have found that bi-level tables are sufficient to decode the bits efficiently. Typical first level table widths are 8 or 9 bits, resulting in 256 or 512 entry tables. Second level tables are much narrower, often 2 or 3 bits.

A simple improvement to multi-table decoding is to decode multiple tokens whenever possible . Since most common codes are short, we may be able to decode multiple tokens much of the time if our tables are sufficiently wide.

3. RESULTS

We have achieved 10 frames/s encode and decode at QCIF resolution with a software implementation of H261.

The encoder timing data in Table 3.1 includes motion estimation and compensation, color transformation, filtering, DCT, quantization, entropy coding, and reconstruction of the decoded frame in the feedback loop. Similarly, decoder timing in the table covers the analogous functions. Performance is affected by data rate as a result of the adaptive transform used. Table 3.2 shows a two fold change in decode speed over the range 64-384 Kbits/s. The encoder is less sensitive and varies from 16 to 12 frames/s over the same range of data rate.

Table 3.2 shows SNR results for a typical talking head sequence at different bit rates. For the case where all blocks are encoded as standard 8x8's, the SNR ranges from 29-33 dB as the data rate varies between 64 -384 kbit/s . Using all 4x4's, which generates noticeably blurred video, results in a 1-3 dB drop. The larger drops occur in high bitrates where 4x4 block types are less likely to occur naturally as a result of quantization. A mixed block type case, where blocks are forced to Zero Blocks, 4x4, DC or 8x8's according to block content, results in subjectively better results with a lower drop in SNR . The block type mix used is shown in Figure 3.1. An interesting side effect of block categorization is improved quality at low data rates. This is probably due to the fact that forcing low energy blocks, which contribute little to SNR, to zero allows bits to be used in other areas where they have more impact on SNR

4. CONCLUSIONS

This paper demonstrates that real-time encoding and decoding of transform based compression is feasible if an adaptive transform is used and motion estimation is performed sparsely. The resultant quality degradation from these com-

Table 2.2a: Percentage of block types in Y channel

kbits/s	8x8	4x4	DC	Zero
64	15.2 %	19.6 %	7.1 %	58.1%
128	43.4	15.4	5.5	35.7
224	67.7	10.5	4.0	17.8
384	84.1	6.2	3.8	5.9

Table 2.2b: Percentage of block types in U,V channels

kbits/s	8x8	4x4	DC	Zero
64	6.7 %	14.7 %	8.1 %	70.4 %
128	18.9	22.7	7.2	51.2
224	36.3	26.2	6.0	31.5
384	63.4	20.8	5.7	10.1

Table 2.2: Percentage of decoder detected 8x8, 4x4, DC, and Zero Blocks. Y is calculated as a fraction of total Y blocks and UV as a it color talking-head of frame size 176x144.

promises is not significant. We believe that this implementation marks an important turning point in the growth of teleconferencing. It is now possible to proliferate this technology quickly without the need for specialized hardware. At present QCIF seems to be the limit in terms of available computation. CIF would appear to require 4 times this capability; however, allowing for some further algorithmic improvements and the fact that CIF frames may contain less information than 4 QCIF frames, it will probably take 2-3 times current capability for full CIF encode and decode at 10 frames/s. With processor speed doubling every 18 months, we expect full CIF performance by mid to late 96.

Hardware based systems will continue to be used for higher performance and in applications where a high performance processor is not already available. Such areas include dedicated videophone and consumer products where integration of functions and high volumes lead to cost efficiencies.

5. REFERENCES

[1] K.S. Wang, J.O. Normile, H. Wu, A.A. Rodríguez, "Vector-quantization-based video codec for software-only playback on personal computers," to appear in Multimedia Systems, Springer-Verlag, Dec. 1994.

[2] J.O. Normile, and D. Wright, "Image compression with coarse grain parallel processing," ICASSP'91, pp. 1121-1124.

[3] CCITT Recommendation H.261, "Video Coding for audiovisual services at px64 kbits/s," Geneva, Aug. 1990.

[4] A.N. Netravali, and B.G. Haskell, Digital Pictures Representation and Compression, Applications of Communications Theory, Plenum Publishing Corporation.

[5] H. Wu, K.S. Wang, J.O. Normile, D.B. Ponceleón, and K. Chu, "Performance of a real-time software-only H.261 codec on the Power Macintosh," to appear in SPIE 95

[6] S.C. Chan and K. L. Ho, "A New Two-Dimensional Fast Cosine Transform Algorithm," IEEE Transactions on Signal Processing, Vol 39 No. 2, pp. 481-485, Feb. 1991.

[8] K. Chu, J.O. Normile, C.L. Yeh, and D. W. Wright, "Variable Length Decoding Using Lookup Tables," U.S. Patent No. 5,235,053, Oct. 1993.

Table 3.1a: Encoder timings in frames/sec

kbits/s	all 8x8	all 4x4	mix
64	16.4	19.2	22.2
128	14.7	16.4	17.5
224	13.3	14.5	18.9
384	12.0	13.9	15.9

Table3.1b: Decoder timings in frames/sec

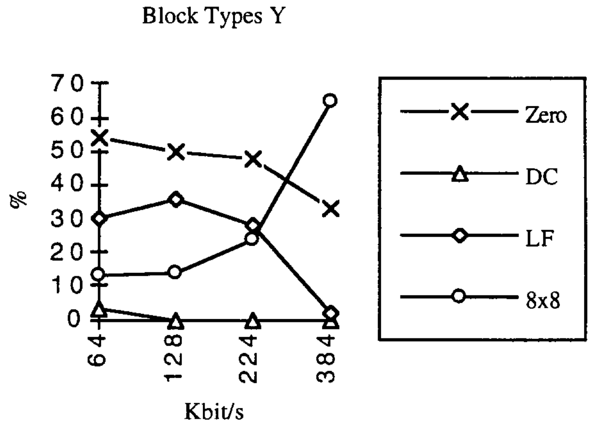
kbits/s	all 8x8	all 4x4	mix
64	45.4	47.6	52.6
128	31.2	28.6	34.5
224	23.8	22.7	35.7
384	20.0	20.8	27.8

Table 3.1: Timing results from a QCIF talking head sequence using a PowerMac 8100/80 (80 MHz 601) running System 7.1. Timing does not include QuickTime managed frame grabbing. Bracketed numbers are in frames/s. The mixed block types ratios are shown in Figure 3.1.

Table 3.2: SNR in dB

kbits/s	all 8x8	all 4x4	mix
64	29.4	28.3	29.0
128	30.7	29.5	30.4
224	31.6	29.9	31.2
384	33.2	30.2	31.9

Table 3.2: Signal to Noise Ratio (SNR) values for different combinations of encoded block types at different datarates. SNR does not include color conversion operation.



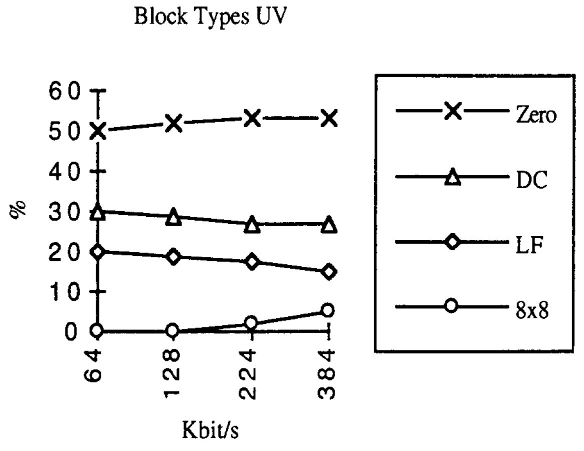


Fig 3.1 Distribution of block types at different bit rates. These reflect the set of thresholds we used to achieve good image quality.