

An Efficient Real Time Low Bit Rate Video Codec

Shikha Tripathi¹, R. Vikas², and R.C. Jain¹

¹Electrical & Electronics Group,
Birla Institute of Technology & Science, Pilani, India
{shikha, rcjain}@bits-pilani.ac.in

²Electronics & Instrumentation Group,
Birla Institute of Technology & Science, Pilani, India
vikas_84bf@rediffmail.com

Abstract. The implementation of a codec for real time applications such as video-conferencing at low bit rates is discussed. The discrete cosine transform has been used for compression, both in two spatial axes as well as in the time axis of the video sequence. A new method of frame by frame processing is proposed which reduces the real time delay associated with processing and transmission of successive video frames, with minimal memory and processing overhead. This implementation is inherently simple and also provides improved performance compared to other popular codecs

1 Introduction

The need for compression is constantly increasing due to the multimedia nature of mobile data. Video compression helps overcome this problem and is a necessary step for widespread introduction of applications of video based mobile phones like teleconferencing and video broadcasting. A satisfactory video compression technique must have the following characteristics:

- It should produce levels of compression comparable to MPEG based and other standards without objectionable artifacts.
- It should be able to compress as well as play back in real time with inexpensive hardware support.
- It should incorporate minimal delay, memory and computational complexity.
- It should not degrade much under network overload or on a slow platform
- It should be resilient to expected types of errors like packet loss during transmission.

The proposed scheme aims at satisfying almost all the above mentioned criteria. A simplified scheme of the compressor with various components is given Fig.1.

In the next section, the scheme, which implements the DCT in the third dimension or time axis, is discussed. In the sections that follow, the mathematical analysis of the proposed algorithm is explained, where a formula for error function has been derived and a method to minimize the cumulative error has been proposed. This is followed by a discussion of the actual results obtained. Other components are based on standard methods [1,3,5].

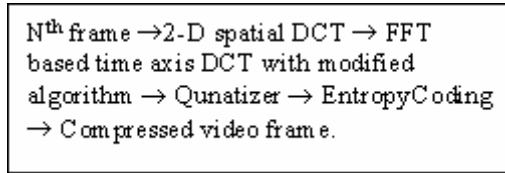


Fig. 1. The Proposed Encoded Structure

2 The 3-D DCT and the Proposed Algorithm

The DCT is popular for its property of compact energy redistribution among fewer components in the transform domain compared to the original image. The formula for the three dimensional DCT (size 8X8X8) is :

$$F(u,v,w) = C(u)C(v)C(w) \sum_{x=0}^7 \sum_{y=0}^7 \sum_{z=0}^7 f(x,y,z) [\cos(2x+1)u\pi/16] * [\cos(2y+1)v\pi/16] [\cos(2z+1)w\pi/16] \quad (1)$$

where x,y,z are pixel indices in pixel space; f(x,y,z) is the value of a pixel in pixel space; u,v,w are pixel indices in DCT space; F(u,v,w) is a translated pixel value in DCT space; C(i)=1/√2 for i=0 and C(i)=1 for i>0.

Although standards like MPEG use different methods like motion compensation, to take advantage of correlation among successive frames, it has been found that such methods are computationally very complex and are therefore not well suited for transmission at low bit rates whenever computational simplicity is desired. Hence, of late, some research has been directed towards pure transform based techniques, which are simpler to implement, the 3-D DCT being one among them. Since an efficient real time fast algorithm for a “true” 3-D DCT has not been found yet, Eq(1) is implemented by running the one dimensional transform once in all three axes: namely the X-Y spatial axes and the temporal axis T. DCT is found to perform optimally for all the three axes if the number of elements is 8[2]. Detailed structure of this algorithm is given in [7]. However, research has also been conducted on adaptive block size based on the estimation of the amount of motion present in the sequence. [6]

In the conventional method using 3-D DCT [1], for every raw uncompressed video frame arriving at the coder, the DCT is first run in the X and Y directions for every 8X8 block in the frame. Then, the temporal DCT is run on each row in the time axis, for 8 such successive frames collected. For example, one of these temporal rows on which the time axis DCT acts is a set of pixels in positions (X=2,Y=3) in frames 1,2,3,4,5,6,7 and 8, which means a pixel in the same spatial location is taken from 8 successive frames and the DCT is run on the row of 8 pixels thus formed along the temporal axis. This is repeated for all such possible rows along time axis. This is equivalent to splitting the whole video sequence of frames into 3 dimensional cubes of size 8X8X8 and running 3-D DCT on all such 3-D cubes formed. However, this conventional method presently used for 3-D DCT implementation [1] that has proved to be much simpler to implement compared to MPEG and motion based techniques, has the following drawbacks:

- The transform in time axis runs on 8 frames at a time, therefore each time it has to wait for 8 frames to accumulate before it can run. This would lead to considerable time delay at the encoder as well as at the decoder.
- This algorithm requires storage of 8 frames at any instance of time, thus requiring considerable memory space.
- Accumulation of 8 frames and their simultaneous transmission after processing requires large bandwidth.

Our proposed algorithm, given below, eliminates these drawbacks and also maintains the simplicity in computation:

Step 1: We are required to wait only for the first frame to accumulate and initialize the sum buffer B_{sum} to 0.

Step 2: Then the intraframe 2-D DCT is run on all 8X8 blocks of the first frame and transmit the data (after quantization and entropy coding).

Step 3: This is repeated for the next 6 frames and the buffer B_{sum} is kept updated, by adding the pixel values of each new frame to B_{sum} . It is to be noted that only the buffer B_{sum} is stored (which is of the size of a single frame) and not the frames themselves, thus saving on memory space.

Step 4: The buffer B_{sum} is stored, which corresponds to the first 7 frames that have been coded. This frame consists of the sum of the values of those pixels that occupy the same spatial position in successive frames. Eg: Sum of pixels in position (1,1) in frames 0 to 6.

Step 5: From frame 7 onwards, for every new frame coming in, the intraframe 2-D DCT is performed on that frame. Then, the temporal axis DCT is run for a group of 2 frames: the most recent frame and the other stored frame (B_{sum}). This is run starting from the most recent frame moving towards the other stored frame (B_{sum}). After this, B_{sum} is updated based on the modified algorithm explained in the next section.

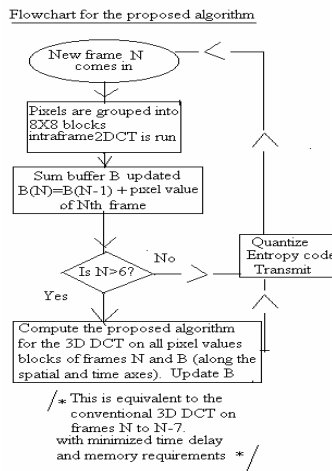


Fig. 2. Flowchart for the proposed algorithm

Step 6: Unlike the conventional 3-D DCT algorithm, which requires storing of 7 previous frames at any time, the proposed algorithm makes use of a single frame storage approach. This frame stored is the buffer B_{sum} .

Step 7: Thus for each new frame N ($N > 6$) which comes in now, use the frame N and buffer B_{sum} to encode this frame N using the 3-D DCT (followed by the standard quantization and entropy coding). This requires a modification of the 3-D DCT, which is explained in the next section.

The flowchart for the proposed algorithm is given in Fig. 2.

3 Modification of the 3-D DCT

The 3-D DCT for each new frame N coming into the encoder (frame 8 onwards) can be expressed in terms of the 2-D intraframe DCT applied on the frame N as well as the temporal axis DCT applied on the Frame N along with 7 other previous frames.

An important difference from the conventional method is that the temporal DCT is applied here in the reverse direction to that of the order of frame arrival, i.e. for frames N to $N-7$, by considering the most recent frame (frame N) as the first frame (frame ‘0’) for the temporal DCT. Since these previous frames are not stored, let only one “Sum” frame S be desired to be stored due to memory constraints, we adapt and modify the 3-D DCT to be applied on these 2 frames S and N only. Note that this frame S , which should contain the sum of pixels from frames N to $N-7$, is different from the actually stored buffer B mentioned earlier. The reason why this difference arises as well as the relation between S and B is derived later in this section. It will be shown that DCT frame ‘0’ (frame N in this case, since we are running the DCT in the reverse direction, from frame N moving towards frames $N-1$, $N-2$ and so on up to frame $N-7$) can be thought of as the DC frame, conveying the information common to each of the image frames. The other DCT frames all convey AC information, which corresponds to motion in the original image sequence. frame ‘0’ (frame N here) is 2-D-Discrete-Cosine-Transformed to produce one frame $F(0)$, with 2-D-DCT coefficients. Next, let all 8 frames (i.e. frame N to frame $N-7$) be 3-D-Discrete-Cosine-Transformed to produce a set of cubes ($8 \times 8 \times 8$ sized) of 8 frames with 3-D-DCT coefficients $F(u,v,w)$. The coefficients $F(0,v,u)$, which correspond to frame ‘0’ (frame N in this case) can be calculated using Eq(1) by setting $w=0$ and rearranging w , y and x axes coefficients.

$$F(0,v,u) = C(u)C(v)C(w) \sum_{x=0}^7 \sum_{y=0}^7 \sum_{z=0}^7 f(x,y,z) [\cos(2x+1)u\pi/16]^* [\cos(2y+1)v\pi/16] [1] \tag{2}$$

The rearrangement is done as it enables the use of the proposed algorithm along the temporal axis. Then, letting original image sequence $F(z,y,x) = F(0,y,x) + \delta(z,y,x)$, the operations on $F(0,y,x)$ and $\delta(z,y,x)$ can be considered separately [2,4]. Here, $F(0,y,x)$ is frame ‘0’ which when transformed, gives the ‘DC’ coefficient frame $F(0,v,u)$ containing the average pixel values for the 8 frame sequence considered. $\delta(z,y,x)$ is the term proportional to the amount of motion present in the frame sequence. We are interested in the transform coefficient set of frame ‘0’ (which is frame N in our frame sequence) $F(0,y,x)$ which is $F(0,v,u)$ and it is explored further below.

$F(0,v,u)$, which is the 3-D DCT coefficient set for frame ‘0’ (frame N) can be decomposed into two components: these components are: 1) a term proportional to $F(0)$ which contains the 2-D DCT coefficients; and 2) a term proportional to the amount of motion present in the 8 image frames (frames N-1 to N-7), relative to the first frame.(frame N).Thus, this gives:

$F(0,v,u) = \sqrt{8} * F(0) + S_{\delta}(v,u)$; Where, the relative motion term,

$$S_{\delta}(v,u) = C(u)C(v)C(w) \sum_{x=0}^7 \sum_{y=0}^7 \sum_{z=0}^7 \delta(x,y,z) [\cos(2x+1)u\pi/16] * [\cos(2y+1)v\pi/16] * [1] \tag{3}$$

Here, $\delta(x,y,z)$ is a pixel value relative to the corresponding pixel in the first frame (frame ‘0’) of the 8 frame sequence, with the same spatial location (x,y). Thus, the values in $S_{\delta}(v,u)$ depends on $\delta(x,y,z)$ for a given X and Y coordinate. Thus, $S_{\delta}(v,u)$ is directly dependent on the relative values of pixels in frames 1 to 7 (N-1 to N-7) with respect to the pixels (with same spatial coordinates) in frame ‘0’ (frame N). Thus for a particular spatial location (x=X,y=Y) and corresponding transform domain location (u=U,v=V),

$$S_{\delta}(v = V, u = U) = KD \tag{4}$$

where,

$$D(x = X, y = Y) = \sum_{z=1}^7 f(X, Y, z) - f(X, Y, 0) \text{ and } K \text{ is a proportionality constant.}$$

$D(X,Y)$ can be re-written as,

$$D(X, Y) = \sum_{z=1}^7 f(X, Y, z) - [7 * f(X, Y, 0)] \tag{5}$$

Thus, for each frame, we require the sum of the 7 previous frames. For the first frame coded in this manner (frame 7), the sum frame $S(x=X,y=Y)$ for a particular spatial location (X,Y) is given by,

$$S_1(X, Y) = \sum_{z=0}^6 f(X, Y, z) - [7 * f(X, Y, 7)] \tag{6}$$

Note that frame ‘0’ for this case is frame 7, which depends on pixel values of previous frame numbers 6, 5, 4, 3, 2, 1 and 0. Similarly, for the second frame encoded in this manner (frame 8),

$$S_2(X, Y) = \sum_{z=1}^7 f(X, Y, z) - [7 * f(X, Y, 8)] \tag{7}$$

Extending this idea to any frame N, let $f(X,Y,z)$ be denoted by $f(z)$ and $S_N(X,Y)$ by S_N . Then, from Eq(7), the desired frame sum S_2 requires the sum $f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7$. However, the sum buffer B_{sum} , which has been maintained in the encoder as stated earlier, actually has the value of buffer sum equal to $B_{sum} = f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7$. It is clearly seen that the buffer B_{sum} contains an extra term f_0 (an unwanted contribution from frame 0) that should be removed. However, the problem is that we no longer have the values of pixels from frame 0. To overcome this problem and still

minimize the effect of the pixels from frame 0 as desired (which is what would have happened had we been able to directly remove the extra term f_0 from B_{sum}), the following procedure is followed:

Subtract the mean M of pixel values f_0 to f_6 from the buffer B_{sum} and store it as B' , instead of storing B_{sum} .

$$B' = B_{sum} - M \tag{8}$$

where $M = (f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6)/7$

This results in an error due to the approximation of desired frame sum $S(X,Y)$ by the actually stored sum buffer value B' , which is:

Error,

$$E_2 = E_N = S(X,Y) - B' = (f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7) - [(f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6) - (f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6)/7] - f_7. \text{ On simplification,}$$

$$E_2 = (f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6)/7 - f_0 \tag{9}$$

Since for a slowly moving video, the changes in the pixel values are gradual, expected value of mean M is f_0 , hence error $E \rightarrow 0$, thus this method is proved to be convergent, which has also been validated with experimental results. Therefore, the approach we follow is to find the Error for frame N , $E_N = S_N(X,Y) - B'$ and try to minimize it. For a frame $N+6$ (run N of this motion based approach after the first 7 frames have been intraframe coded), E_{N+6} can be minimized by considering another related value C , which is defined as the error due to the unwanted contribution of a frame L ($L < N+6$). As an example, $C_{N+6}(0)$, which is the error due to the unwanted contribution of first frame, frame 0 to the above described processing of frame $N+6$, expressed as a percentage of the pixel values in frame 0 is obtained by extending Eq(8) and Eq(9) for any value N , and is given by,

$$C_{N+6}(0) = [1/7 + 1/8 + + 1/(N + 5) - 1] * 100 \tag{10}$$

Similarly, percent error due to contribution of pixels of frame 1 in the processing of frame $N+6$ is,

$$C_{N+6}(1) = [1/8 + 1/9 + + 1/(N + 5) - 1] * 100 \tag{11}$$

and so on. We design the receiver structure in order to minimize this cumulative error. Extending Eq(10) and Eq(11) for any general frame value L ($L < N+6$), the total cumulative error sum due to $C(0)$, $C(1)$ etc. totally affecting a transmitted (and hence received frame) $N+6$, is given by a recursive relation,

$$Q_{N+6} = Q_{N+5} + \left[\sum_{i=0}^{N+4} f_i / (N + 5) \right] - f_{N-2} \tag{12}$$

Thus, at the receiver, we need to have a Q buffer, which we keep on updating by adding Q_N whenever a new frame received is decoded. This buffer Q thus contains the cumulative error history which takes care of the (recursive) first term in the right hand side of Eq(12). This value can be subtracted from the received coefficient values

to remove first error term. Also, similar to the transmitter, we also have a received frames sum buffer R , which is constantly updated by adding the scaled values (i.e. multiplied by $N+5$) of the pixels in the latest frames decoded. This removes the second error term in the right hand side of Eq(12). The last term is of concern since this requires the storage of a frame that has been decoded 8 frames before the present frame. This requires a buffer H of the most recent 8 frames at the receiver, which is updated constantly, which is the main memory requirement for the decoder, similar to the 3-D DCT decoder. If this is done, the third term in the right hand side of Eq(12) is also eliminated thus removing all error terms.

4 Results

The encoder and decoder were simulated using MATLAB 6.1. The simulation was carried out for the conventional 3-D DCT, the proposed codec, and also the intraframe 2-D DCT based codec. The original and encoded/decoded frames from a real time captured video are shown in Fig. 3. and Fig. 4. The calculation of the PSNR is in Table. 1.

Although not evident here, there is a fair degradation of quality in this case, unlike the next sequence “Seaplane” where the quality is not lost much. This is because the original “Seaplane” sequence is of medium quality.

Also, Table. 1 indicates that even before quantization and entropy coding, the bits per pixel value of 0.7 gives a PSNR of 15 to 24 dB that is acceptable for real time streaming. Further, in Table. 2, the comparison of the proposed method with the conventional 3-D DCT is given.

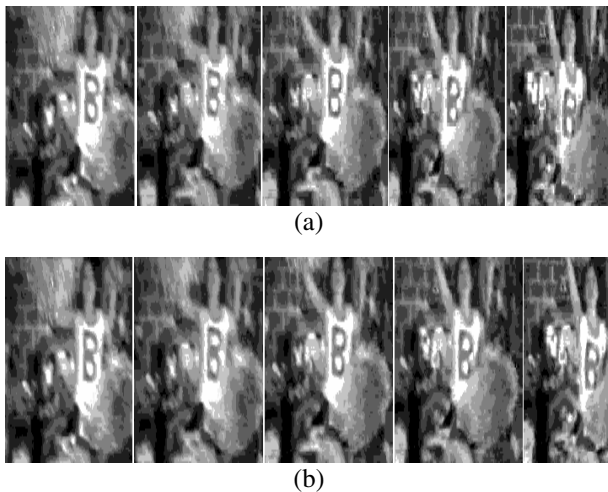


Fig. 3. (a) “Girl”-Original sequence (noisy in nature, typical of low quality streaming video)at 8 bits/pixel. (b) “Girl” sequence passed through the codec at 0.7 bits/pixel.



(a)



(b)

Fig. 4. (a) “Seaplane” at 8 bits/pixel (A medium quality video). (b) “Seaplane” after compression to 0.7 bits/pixel. The medium video quality is maintained even after compression in this case.

Table 1. Statistics of the 2 video sequences

Sequence	Bits per pixel (b.p.p)	PSNR(dB) (proposed method)	PSNR(dB) (conventional method)
1.Girl (noisy/streaming quality)	0.7	15.10	15.94
2.Seaplane (medium quality)	0.7	23.66	24.49

Table 2. Comparison between conventional 3-D DCT, simple intraframe 2-D DCT & proposed method for various test video sequences

Method of compression used	Time Delay in frame accumulation (in seconds)	Encoder Memory requirement (for a 340*240 sized video sequence) in kilobytes	Bits per pixel (b.p.p) (before quantization and entropy coding)
Conventional 3-D DCT	0.25	614.4	0.7
Proposed algorithm	0.03	153.6	0.7
Intraframe 2-D DCT only (for each frame)	0.03	76.8	1.875

For completeness, simple intraframe 2-D DCT has also been considered (which is similar to motion JPEG and does not take advantage of the temporal correlation among frames). It is clear from Table. 2 that the proposed algorithm retains the higher compression property of 3-D DCT. However, at the same time it does away with the time delay by reducing it to nearly (1/8)th of the delay in the 3-D DCT method. Additionally, the proposed method has the advantage of requiring lesser memory, which is (1/4)th that of 3-D DCT and is almost comparable to the minimal memory requirements of the basic 2-D DCT method.

5 Conclusion

The proposed algorithm has a performance, which is considerably superior to the conventional algorithm. In terms of time, the improvement is almost of an order of magnitude and requirement of memory is only about 25% of that required for the conventional method. These advantages are achieved while still retaining almost the same video quality as that obtained by the conventional 3-D DCT. As shown in the mathematical analysis in section-4, this method is convergent leading to zero error. However, there is scope for further improvement in finding the function, which minimizes the buffer H memory requirement at the decoder further over a large range of frame number N.

References

1. R. Westwater, B. Furht.: Real-Time Video Compression. Techniques and Algorithms. Kluwer Academic Publishers. Boston (1997)
2. M Servais, G Jager.: Video Compression using the 3 dimensional Discrete Cosine Transform. Proceedings of IEEE, COMSIG '97 (1997) 27-32.
3. Video Coding for Low Bit Rate Communication, H.263 Standard, ITU-T Recommendation H.263, February (1998)
4. B.P.Lathi.: Signal Processing and Linear Systems. Oxford University Press (1998)
5. P. Clarkson, H. Stark(Ed):. Signal Processing Applications for Audio, Images and Telecommunications. 2nd Edition. Academic Press (1995)
6. Furht, B., Gustafson, K., Huang, H., and Marques, O.: An adaptive three-dimensional DCT compression based on motion analysis. In Proceedings of the 2003 ACM Symposium on Applied Computing. ACM Press, New York. (2003) 765-768
7. Xiuqi Li and Borko Furht.: An Approach to Image Compression Using Three-Dimensional DCT. Proceedings of the Visual 2003 Conference. Miami, Florida. (2003)