An Efficient Arbitrary Downsizing Algorithm for Video Transcoding

Haiyan Shu and Lap-Pui Chau

Abstract—When delivering video over communication network, it is required to transcode the pre-coded video content to meet the demands of a broad range of end users with different bandwidth and resource constraints. One solution to transmit video over bandwidth-constrained channels is to reduce the spatial resolution of video frame and transmit a low-resolution version of video as a tradeoff for the bit-rate. In this paper, an arbitrary downsizing algorithm is proposed. This arbitrary downsizing algorithm is processed directly in the DCT domain. Experimental results show that the proposed method can achieve a satisfied performance. Compared with the existing methods, this algorithm is not only applicable to intra coding frame but also inter coding frame. This is one of the crucial features of this method.

Index Terms—Video transcoding, Arbitrary downsizing.

I. INTRODUCTION

Today, the Internet brings heterogeneous devices together. Different end users may have different network specifications, and consequently, different demands for video transmission bit rate and quality. The original video source may be first pre-coded at higher resolution and higher bit rate. To match the bit rate of the video source to the channel constraints and the display screen of the end devices, the spatial resolution of the original video needs to be lowered as a tradeoff for the bit rate. Transcoding has been adopted as a very promising technique to realize lowering the video resolution. The traditional and expensive approach would be downsizing the video frame in spatial domain [8]: first fully decode the compressed video stream, and then apply filter techniques to downsize the decoded video, finally, encode the video sequence again. This straightforward approach is undesired due to the significant computational overhead associated with decoding and encoding as well as the large memory requirement. Therefore, it is extremely worthwhile to develop fast algorithms to reduce the computation.

Now many approaches proposed to achieve this in the compressed domain. By using the distributive character of unitary orthogonal transform, Chang [1] proposed to manipulate image scaling in DCT domain. Some researchers have proposed some algorithms that can realize image downsizing in DCT domain [2]-[5]. But all these approaches can only achieve downsizing by a factor of two, or the downsizing ratio must be integer. In practice, due to the versatility of the display devices, an arbitrary downsizing ratio is expected. Recently, Mehta and Desai [6] have proposed an arbitrary image downsizing method. By converting \( m^n \) neighboring blocks of size \( 8 \times 8 \) into \( n^2 \) blocks of size \( 8 \times 8 \), this method can realize image resizing in ratio \( m/n \). This method is computational expensive due to the large size of matrix multiplication. Besides, it can only process still image, where in practice, video stream is in great need to be processed.

In this paper, we propose an algorithm that can achieve arbitrary image/video downsizing. This algorithm takes the advantage of compressed domain processing techniques and is processed completely in DCT domain without introducing further computation. When combined with the transcoding method mentioned in [7], which can estimate the motion vectors from the input bit stream for arbitrary downscaled video, our proposed method can efficiently process video stream downsizing. Compared with the existing methods, this algorithm is not only applicable to intra coding frame but also inter coding frame.

The paper is organized as follows. In section 2, we describe the proposed arbitrary downsizing algorithm in detail. Experimental results will be presented in section 3, and conclusion is drawn in section 4.

II. DERIVATION OF ALGORITHM

In spatial domain, for an arbitrary downsizing ratio \( R \), which is defined as the ratio of original resolution to the desired resolution (note that the horizontal downsizing ratio \( R_x \) can differ from the vertical downsizing ratio \( R_y \)), more than one pixel in the original frame may contribute to one single pixel in the downsized frame.

As shown in Fig. 1, one \( 8 \times 8 \) output block in the downsized frame can come from as many as \( M \times N \) related blocks \( a_{y,b} \), which the supporting area \( b \) in size of…
$8R_x \times 8R_y$ may cover in the original frame. The original frame can be partitioned into these supporting areas. As we can see, for arbitrary downsizing, these supporting areas may not align to the block border. Then our approach can realize the downsizing in two steps: extracting the supporting area from the original frame, and then downsizing it into $8 \times 8$ output block.

A. Extracting supporting area in the original frame

Due to the non-integer downsizing ratio, some related blocks in the original frame might partially contribute to certain output block in the downsized frame. As shown in Fig. 2, these related blocks could be classified into 9 different cases based on covered locations. They can be totally covered or partially covered by the supporting area. The partially covered blocks can be further classified into 8 cases, where the overlapped regions locate at the top, bottom, left, right, top left, bottom right, left, and bottom right portions of the block in the original frame, respectively.

In the scenario of integer downsizing, all pixels of the related blocks in the original frame are used to form the output block in the downsized frame. However, when the downsizing ratio is not an integer, not all pixels of these related blocks would be engaged in generating the output block. For instance, if the supporting area covers the bottom right corner of the related block, e.g. case 1 in the Fig. 2, only a portion of the pixels in this block (bottom right $p \times m$ pixels as shown in Fig. 2) are used to form the new block. These cases may partially contribute to the whole output block in the downsized frame.

Assuming that we have already partially decoded these related blocks $a_{ij}$s, we wish to extract the spatial information of the supporting area $b$ from $A_{ij}$s, where

$$A_{ij} = DCT(a_{ij}) \quad i=1 \text{ to } M, j=1 \text{ to } N$$ (1)

Spatial domain representation can serve as the starting point to build relation between $b$ and $A_{ij}$s.

We first extract those related pixels that constitute the supporting area from $a_{ij}$s. We can get $M \times N$ blocks namely $\hat{a}_{ij}$. The size of $\hat{a}_{ij}$s may differ and depend on the covered pixels of each related block. Let’s use an example to illustrate our ideal: for case 6 in Fig. 2, only the left $q \times 8$ pixels are covered by the supporting area $b$, then the size of $\hat{a}_{ij}$ is equal to $q \times 8$. Next we consider another $M \times N$ blocks of size $8R_x \times 8R_y$ namely $\bar{a}_{ij}$. Here $\bar{a}_{ij}$s are the zero padding versions of $\hat{a}_{ij}$s and can be obtained from $a_{ij}$s in the following:

$$\bar{a}_{ij} = m_{ijL} \cdot a_{ij} \cdot m_{ijR}$$ (2)

where $m_{ijL}$ and $m_{ijR}$ are shift matrices used to extract the corresponding matrix $\hat{a}_{ij}$ from the related block $a_{ij}$ and pad it into the size of $8R_x \times 8R_y$. They are in size of $8 \times 8R_x$ and $8 \times 8R_y$ respectively.

Then the supporting area $b$ can be represented by combining $\bar{a}_{ij}$s in the following:

$$b = \sum_{i=1,j=1}^{M,N} \bar{a}_{ij}$$

$$= \begin{pmatrix} \hat{a}_{11} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{a}_{1j} \\ \hat{a}_{21} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{a}_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{a}_{ij} \\ \hat{a}_{M1} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{a}_{Mj} \\ \hat{a}_{MN} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{a}_{MN} \end{pmatrix} + \begin{pmatrix} 0 & 0 \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 \cdots & 0 \end{pmatrix}$$ (3)
Due to the distributive property of the DCT, we can get the following relation:

$$
\overline{A}_y = DCT(\overline{a}_y) = M_{yL} \cdot A_y \cdot M_{yR}
$$

(4)

$$
B = DCT(b) = \sum_{i=1}^{M} \sum_{j=1}^{N} \overline{A}_j
$$

(5)

where $M_{yL}$ and $M_{yR}$ are the DCT representations of $m_{yL}$ and $m_{yR}$ respectively.

At last, we can extract the supporting area $b$ in DCT domain, which is defined as follows:

$$
B = \sum_{i=1}^{M} \sum_{j=1}^{N} M_{yL} \cdot A_y \cdot M_{yR}
$$

(6)

B. Downsizing the supporting area

In natural image, most of the signal energy is concentrated in the lower frequency part in DCT domain. A reasonable downsizing scheme as proposed in [4] is to retain only the lower frequency component and discard the high frequency components of the block. Most of the energy of the original block is preserved. As we can see above, the size of supporting area is $8R_x \times 8R_y$. This supporting area may contribute to one $8 \times 8$ output block. We need to discard the high frequency component and extract only the low frequency part of size $8 \times 8$ to downsize the block $B$ to $8 \times 8$ in DCT domain.

$$
\hat{B} = [I_8 \ 0]_{8 \times 8R_x} \cdot B \cdot [I_8 \ 0]_{8R_y \times 8}
$$

(7)

The block $\hat{B}$ is a $8 \times 8$ DCT block. No further processing is needed. At last, we can get the relation between $A_y$'s and $\hat{B}$ in (8), and downsizing is realized from $8R_x \times 8R_y$ block to $8 \times 8$ block in DCT domain.

$$
\hat{B} = \sum_{i=1}^{M} \sum_{j=1}^{N} H_{yL} \cdot A_y \cdot H_{yR}
$$

$$
= \sum_{i=1}^{M} \sum_{j=1}^{N} [I_8 \ 0]_{8 \times 8R_x} \cdot M_{yL} \cdot A_y \cdot M_{yR} \cdot [I_8 \ 0]_{8 \times 8R_y}
$$

(8)

As we can see, both pre-matrix $H_{yL}$ and post-matrix $H_{yR}$ are of size $8 \times 8$. These matrices are independent of the input blocks so that all possible combinations of $H_{yL}$ and $H_{yR}$ can be pre-computed and stored in the memory. With look-up-table based implementation method, no delay is imposed while processing real-time video transcoding. The purpose of conducting the operations solely in DCT domain is to reduce the computation complexity. Compared to the conventional scheme that processing in spatial domain, the DCT and IDCT are avoided. In the method proposed in [6], the size of pre-matrix and post-matrix are $8m \times 8$ and $8 \times 8m$ in blocks merging and $8 \times 8n$ and $8n \times 8$ in blocks splitting. For large $m$ and $n$, the multiplication will increase significantly. Where in our approach, both of the pre-matrix $H_{yL}$ and post-matrix $H_{yR}$ are in size of $8 \times 8$. The matrix size is smaller than the method proposed by [6] and hence the computation can be saved.

III. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, experimental results are presented for the proposed arbitrary downsizing algorithm. The original frame size is $352 \times 288$. This frame is downsized to size of $256 \times 192$, where the horizontal downsizing ratio $R_x$ is $11:8$ and vertical downsizing ratio $R_y$ is $3:2$ respectively.

Frame 40 from video sequence “coastguard” is shown in Fig. 3. In the experiments, downsized pictures are upsized to original frame size and compared with the original picture. For our proposed method, downsized picture is up-sampled to original size by DCT domain zero padding. Fig. 3(b) shows the picture downsized by our proposed method. The PSNR is 34.34dB. Fig. 3(c) and Fig. 3(d) show the pictures downsized by the spatial domain bilinear interpolation. The downsized pictures are up-sampled by spatial domain bilinear interpolation and DCT domain zero padding, respectively. The PSNR is 28.80dB by using spatial domain bilinear interpolation upsizing and 30.85dB by using DCT domain zero padding upsizing. This zero padding process introduces no quality enhancement to the picture. Comparison shows that our proposed DCT domain arbitrary downsizing method can well preserve the most important information of picture and present a better visual quality. Less information is lost in downsizing process.

Fig. 4 shows the PSNR comparison of first 100 frames for the video sequence “coastguard” by different downsizing approaches. It is obvious that our proposed method presents a better PSNR than the method proposed in [6]. These two DCT domain downsizing schemes outperform the spatial domain bilinear interpolation method and produce PSNR improvement.

Some other video sequences are tested by our proposed method. The original 100 frames are downsized from size of $352 \times 288$ to size of $256 \times 192$. Downsizing results are shown in Table 1, which shows that our proposed method outperforms the spatial domain processing.

Comparison results show that our proposed DCT domain arbitrary downsizing can well preserve the most important information of the picture, and hence present a better visual quality. Less information is lost in downsizing process. This is one reason that DCT domain downsizing processing is adopted in transcoding process.

When compared the computation complexity of our
proposed method with spatial domain method, the
computation can be reduced by our proposed method. First,
DCT and inverse DCT can be saved by using DCT domain
downsizing. In order to obtain a high quality downsized video
in spatial domain, long filter length for ideal low pass filter
should be applied to downsize picture, but the computation
will increase depending on the required visual quality.
Especially for arbitrary downsizing, the filter matrix may be
less sparse. This also involves a complicated matrix
multiplication, which increases the computation cost. In this
way, it is worthwhile to downsize it in DCT domain for the
better picture quality of DCT domain downsizing.

By further utilizing the share information algorithm
proposed in [9], the computation can be further reduced. Some
other approaches [2, 10] that reduce the computation on
matrix multiplication can also be adopted in our proposed
method to reduce the computation.

As for inter coding frame, frame can be downsized in the
same way. Our approach proposes a partition method in the
original frame. It is a block-by-block processing. In [7], a
transcoding method that can re-estimate the motion vectors
from input video stream for arbitrary size downscaled video
is proposed. In this method, the original frame is partitioned in
the similar way with our approach. Combined our proposed
approach with the method proposed in [7], inter coding frame
could be arbitrary downsized in DCT domain and the motion
vector could be re-estimated from the original motion vector
without re-generating it. This shows that our approach can be
used to downsize not only intra coding frame, but also inter
coding frame. Moreover, the video transcoding can be realized
completely in DCT domain.

IV. CONCLUSION

In this paper, an efficient algorithm, which can downsize
the video frame to arbitrary size, is proposed. This algorithm
operates directly in DCT domain and does not require computing forward and inverse DCT.

Experimental results show that the arbitrary downsizing algorithm we proposed could yield better visual quality. Compared to other algorithms, which operate in the DCT domain itself, our algorithm differs in that it can produce downsizing for arbitrary ratio. This is very crucial for achieving different bit-rate requirements for video transcoding.

This method, when combined with the method proposed in [7], can process not only the still image, but also the video sequence. This is another crucial feature of this method.

REFERENCES


Fig. 4. The PSNR performance of different downsizing methods on “Coastguard”. (Bilinear interpolation downsizing is upsized by bilinear interpolation (1) and DCT domain zero padding (2), respectively.)