FRACTIONAL SCALING OF IMAGE AND VIDEO IN DCT DOMAIN

Yunlong Zhao Mohan S Kankanhalli Tat-Seng Chua

School of Computing
National University of Singapore, Singapore 117543
{zhaoyl, mohan, chuats}@comp.nus.edu.sg

ABSTRACT

We present an algorithm for scaling image and video with fractional factors of 1.25 and 1.50 directly in compressed (DCT) domain without explicit decompression and recompression. It differs from those compressed-domain sampling methods that work only with integer factors, and provides more flexibility in changing image sizes. We employ a simple, consistent and extensible way to implement the algorithm. The resulting scheme ensures that the compressed domain algorithms always use fewer arithmetic operations than their conventional spatial domain counterparts. Experimental results in terms of visual quality and objective evaluation are provided.

1. INTRODUCTION

There are huge amounts of images and videos compressed with JPEG, MPEGs and H.26x. Normally, the image and video must be decompressed before any manipulation and processing operations can be applied to them. After the operations, it is quite common that the processed image and video need to be transformed to compressed format again for storage or transmission purposes. Thus a lot of computation cost can be saved if we can bypass the decompression and recompression process. Therefore, a number of algorithms have been developed for manipulating and processing image and video directly in compressed domain [1, 2, 3, 4, 5].

Among the list of operations, changing the resolution of image or video frame is often necessary. But most of the existing DCT-domain methods support only integer scaling factors [3, 4, 5]. They give users little choice in scaling and the image size tends to shrink or grow too drastically. This severely restricts the wide adoption of scaling algorithm since many applications, such as the detection of variable sized faces with a fixed-sized face model, require that the image or video frame to be scaled by fractional factors ranging between 1 and 2.

Taking the above issues into consideration, we propose a common scheme to perform flexible scaling of image and video by fractional scaling factors of 1.50 and 1.25 in compressed (DCT) domain. It intends to fill the gaps between the possible application requirements and the currently available technologies to support a wide variety of applications. It is important to support applications such as the compressed domain face detection [6]. The only requirement in applications is that the image and video should be organized in 8×8 pixels block structure and in the compressed DCT format, which is widely adopted by JPEG, MPEGs and H.26x. For the sake of simplicity in implementation, we decompose the DCT computation processes into several common steps, and perform these steps sequentially in the vertical and horizontal directions. This gives rise to a simple and fast algorithm that can be easily implemented in both software and hardware.

In Section 2, we discuss related issues and our approach to solve the problems. We then describe the implementation of fractional scaling in Section 3. In Section 4, we present the simulation results and evaluations in terms of subjective and PSNR quality of images obtained by downsampling and upsampling using our method. Finally, we conclude the paper in Section 5.

2. SCHEME FOR DCT-DOMAIN OPERATIONS

As illustrated in Figure 1, the content of a resulting DCT block, depicted as D_{ij}, corresponds to a block in the original image, depicted as D'_{ij}. We need to map each pixel in D_{ij} back to pixels in the original blocks from which the resulting pixels value are predicted. This is usually dependent on at most 4 original neighboring DCT blocks in the source image or video frame, which are B_{ij}, $B_{i(j+1)}$, $B_{(i+1)j}$, $B_{(i+1)(j+1)}$. D_{ij} can be obtained as follows [3],

$$\begin{aligned} D_{ij} &= S^t_{2i} B_{ij} S_{2j} + S^t_{2i} B_{i(j+1)} S_{2j+1} \\ &+ S^t_{2i+1} B_{(i+1)j} S_{2j} + S^t_{2i+1} B_{(i+1)(j+1)} S_{2j+1} \end{aligned} \quad (1)$$

where the superscript t denotes matrix transposition and S_k's are 8×8 matrices that complete the pixel selection and value interpolation. The selection and design of S_k's determine the function of the manipulation operation.

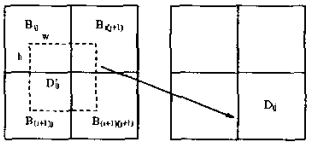

Fig. 1. A typical approach to deriving a resulting DCT block from 4 neighboring DCT blocks in the source image or video.

Based on the distributive property of matrix multiplication with respect to the DCT [3], Equation 1 can be converted to its DCT domain representation as follows,

$$\begin{aligned} \mathbf{D}_{ij} &= \mathbf{S}^t_{2i} \mathbf{B}_{ij} \mathbf{S}_{2j} + \mathbf{S}^t_{2i} \mathbf{B}_{i(j+1)} \mathbf{S}_{2j+1} \\ &+ \mathbf{S}^t_{2i+1} \mathbf{B}_{(i+1)j} \mathbf{S}_{2j} + \mathbf{S}^t_{2i+1} \mathbf{B}_{(i+1)(j+1)} \mathbf{S}_{2j+1} \\ &= [\mathbf{S}^t_{2i} \mathbf{B}_{ij} + \mathbf{S}^t_{2i+1} \mathbf{B}_{(i+1)j}] \mathbf{S}_{2j} \\ &+ [\mathbf{S}^t_{2i} \mathbf{B}_{i(j+1)} + \mathbf{S}^t_{2i+1} \mathbf{B}_{(i+1)(j+1)}] \mathbf{S}_{2j+1} \end{aligned} \quad (2)$$

0-7803-7750-8/03/$17.00 ©2003 IEEE

$$\begin{aligned}\mathbf{D}_{ij} &= T\{[S_{2i}^t R B_2^t B_1^t P^t D \mathbf{B}_{ij} + S_{2i+1}^t R B_2^t B_1^t P^t D \mathbf{B}_{(i+1)j}] D P B_1 B_2 R^t S_{2j} \\ &\quad + [S_{2i}^t R B_2^t B_1^t P^t D \mathbf{B}_{i(j+1)} + S_{2i+1}^t R B_2^t B_1^t P^t D \mathbf{B}_{(i+1)(j+1)}] D P B_1 B_2 R^t S_{2j+1}\} T^t \end{aligned} \quad (3)$$

Equation 2 shows that the derivation of $\mathbf{D}_{ij}$ can be divided into two consecutive steps. First, we process the source blocks vertically by pre-multiplying matrices $\mathbf{S}_k^t$'s to $\mathbf{B}_{ij}$, $\mathbf{B}_{i(j+1)}$, $\mathbf{B}_{(i+1)j}$ and $\mathbf{B}_{(i+1)(j+1)}$ respectively. Second, we process the intermediate results horizontally by post-multiplying matrices $\mathbf{S}_k$'s to obtain $\mathbf{D}_{ij}$. The directional processing is widely employed in image and video processing, and also adopted in compressed domain transcoding of MPEG [7]. This simple computation strategy permits the reuse of computed results and guarantees the simplicity and consistency of implementing the operations under discussion.

2.1. Implementing the Fast Algorithm

To compute matrix multiplications like $\mathbf{S}_{ij}^t \mathbf{B}_{ij}$ and $\mathbf{B}_{ij} \mathbf{S}_{ij}$, we adopt a fast algorithm attributed to Arai et al. [8] and illustrated in detail in [9] and [4]. The fast algorithm is based on the factorization of the DCT transformation matrix T of the form:

$$T = DPB_1 B_2 M A_1 A_2 A_3$$

where D is a diagonal matrix that contains the constant scaling terms; P is a permutation matrix; A_1, A_2, A_3, B_1 and B_2 are sparse matrices with entries 1 and -1; and M is a sparse matrix with entries 1 and four other real vales. Please refer to [9] and [4] for the exact entry values of the matrices.

A fixed matrix $R = (MA1A2A3)^t$ can be pre-computed as follows:

$$R = \begin{pmatrix} 1 & 1 & a & 1 & -b & 0 & c & 1 \\ 1 & -1 & a & 0 & -b & a & c & 0 \\ 1 & -1 & -a & 0 & c & a & b & 0 \\ 1 & 1 & -a & -1 & c & 0 & b & 0 \\ 1 & 1 & -a & -1 & -c & 0 & -b & 0 \\ 1 & -1 & -a & 0 & -c & -a & -b & 0 \\ 1 & -1 & a & 0 & b & -a & -c & 0 \\ 1 & 1 & a & 1 & b & 0 & -c & -1 \end{pmatrix}$$

In order to evaluate the efficiency of this algorithm, we enumerate the computation cost in terms of the basic arithmetic operations proposed by Merhav and Bhaskaran [4]. The elementary arithmetic operations here refer to either an ADD, SHIFT, or SHIFT-ADD operations in the PA-RISC processor. This evaluation method was also adopted by Song and Yeo [10]. We now measure the computation cost incurred by multiplication as shown in Equation 3.

Since the computation cost related to operator S_k depends on the specific design of S_k for a particular manipulation operation, we leave it until we discuss each operation in details later. The multiplication of D can be absorbed by the modified de-quantizer and P is a permutation matrix, we can thus ignore them when we count the number of operations involved. It is obvious that pre-multiplication with $B_2^t B_1^t$ takes $8 \times 8 = 64$ additions and post-multiplication with $B_1 B_2$ also requires $8 \times 8 = 64$ additions.

The computation of $\vec{u} = R\vec{v}$, where $\vec{u} = (u_1, u_2, \cdots, u_8)^t$ and $\vec{v} = (v_1, v_2, \cdots, v_8)^t$, can be carried out in the following steps:

$y_1 = v_1 + v_2 \quad y_2 = v_1 - v_2 \quad y_3 = av_3 \quad y_4 = v_3 + v_4$

$y_5 = b(v_5 + v_7) \quad y_6 = av_6 \quad y_7 = y_5 - (b + c)v_7$

$y_8 = y_5 + (c - b)v_5 \quad y_9 = y_1 + y_4 \quad y_{10} = y_2 + y_3$

$y_{11} = y_2 - y_3 \quad y_{12} = y_1 - y_4 \quad y_{13} = y_6 - y_7 \quad y_{14} = v_8 - y_7$

$y_{15} = y_6 + y_8 \quad u_1 = y_9 + y_{14} \quad u_2 = y_{10} + y_{13}$

$u_3 = y_{11} + y_{15} \quad u_4 = y_{12} + y_8 \quad u_5 = y_{12} - y_8$

$u_6 = y_{11} - y_{15} \quad u_7 = y_{10} - y_{13} \quad u_8 = y_9 + y_{14}$

The total number of operations required is 5 multiplications and 21 additions. Thus, the number of operations for pre-multiplication with R or post-multiplication with R^t is $5 \times 8 = 40$ multiplication and $29 \times 8 = 232$ additions, which is equivalent to $50 \times 8 = 400$ elementary operations in the PA-RISC processor according to [4] and [10]. The operation of DCT needs 672 elementary operations.

3. IMPLEMENTATION OF FRACTIONAL SCALING

In this section, we describe the implementation of DCT-domain downsampling and upsampling by factors of 1.50 and 1.25. This involves the designing of matrices S_i's in Equation 3. A detailed illustration of 1.50 case is given as an example here. Generally, with the fractional scaling factor, every $M \times M$ blocks can be resized to $N \times N$ blocks; and we only need m multiplication matrices as well as their transpositions to implement the scaling in the DCT domain. Here, M equals to 3, N equals 2 and m equals to 4 for the fractional scaling factor of 1.50; and M equals to 5, N equals 4 and m equals to 8 for the fractional scaling factor of 1.25.

Normally, to reduce aliasing effects on the downsampled image, a low-pass filtering must be applied before the downsampling operation. For simplicity, the result after applying an low-pass filtering in both vertical and horizontal directions on the 8×8 DCT block is to keep the top-left $w \times w$ ($w < 8$) coefficients and zeroing all the other higher frequency coefficients [11, 12, 2]. Strictly speaking, this intra-block operation is not equivalent to a sharp low-pass filter in the Fourier domain. Thus many algorithms have been developed using variants of Fourier transform, but they normally require the doubling of the input data or inter-block operations [12, 2]. But the subsequent computational cost tends to weaken the advantages of compressed domain processing. Our approach to the problem is to employ intra-block filtering to preserve the block structure. The essence is to keep the alias introduced tolerable and maintain reasonable trade-off between performance and computational efficiency. Specifically, we set $w = 5$ for the scaling factor of 1.50, and $w = 6$ for 1.25, respectively.

3.1. Downsampling by a Factor of 1.50

The matrices for scaling by 1.50 are all very sparse, with only 3 different entry values, 0, 0.5, and 1. The entry values reflect the role of these matrices to perform pixel mapping and interpolation. The multiplication matrices S_k (k=0,1,2,3) are defined as follows (here we only list the nonzero entry values s_{ij}, where i,j=1-8).

$$\begin{aligned} S_0 &= \{s_{ij} | s_{11} = 1, s_{22} = 0.5, s_{32} = 0.5, s_{43} = 1, \\ &\quad s_{64} = 0.5, s_{75} = 1, s_{86} = 0.5\} \\ S_1 &= \{s_{ij} | s_{16} = 0.5, s_{27} = 1, s_{38} = 0.5, s_{48} = 0.5\} \end{aligned}$$

$$S_2 = \{s_{ij} | s_{51} = 1, s_{62} = 0.5, s_{72} = 0.5, s_{83} = 1\}$$
$$S_3 = \{s_{ij} | s_{14} = 0.5, s_{24} = 0.5, s_{35} = 1, s_{46} = 0.5, s_{56} = 0.5, s_{67} = 1, s_{78} = 0.5, s_{88} = 0.5\}$$

The nonzero entry values in S_k's are 0.5 and 1. Their common factor of 0.5 can be extracted and absorbed in the modified de-quantizer. Then the nonzero entry values in S_k's become 1 and 2. We avoid doing matrix summations in Equation 3 since the summands S_{2i} and S_{2i+1} have non-zero entry values on disjoint subsets of rows.

According to Equations 3, to convert the 3×3 blocks $\mathbf{B}_{ij}$'s to 2×2 blocks $\mathbf{D}_{ij}$'s by down-sampling with a factor of 1.50, the following computations are involved in terms of the elementary arithmetic operations:

1. The filtered source of 3×3 DCT blocks $\mathbf{B}_{ij}$'s are converted to 3×3 blocks by performing matrix pre-multiplication with $RB_2^t B_1^t$. This involves 9 such operations amounting to $220 \times 9 = 1,980$ elementary operations.
2. We next perform the vertical block interpolation to convert the 3×3 blocks to 2×3 blocks. This involves 6 pre-multiplication operations with S_{2i}^t and S_{2i+1}^t, which equals to $40 \times 6 = 240$ elementary operations.
3. The resulting 2×3 blocks are then post-multiplied by $B_1 B_2 R^t$. This involves 6 such operations equivalent to $352 \times 6 = 2,112$ elementary operations.
4. We then perform the horizontal block interpolation to convert the 2×3 blocks to 2×2 blocks. This involves 4 post-multiplication operations with S_{2j} and S_{2j+1}, which is equivalent to $64 \times 4 = 256$ elementary operations.
5. Finally, we derive $\mathbf{D}_{ij}$ by performing 2×2 2-D DCT transforms on the resulting 2×2 blocks. This process requires $672 \times 4 = 2,688$ elementary operations.

Thus to get a block in the resulting image down-scaled by a factor of 1.50 using the present algorithm in the PA-RISC processor, the number of elementary operations needed is: $(1980+240+2112+256+2688)/4 = 1819$. On the other hand, the brute-force method of performing fast IDCT first, down-scaling the image by 1.50 in spatial domain and performing fast DCT again requires 2884 operations. This means a saving of 37% of operations by using our compressed domain down-sampling method.

3.2. Downsampling by a Factor of 1.25

The matrices for scaling by 1.25 are also very sparse, with only 5 different entry values, 0, 0.25, 0.5, 0.75 and 1. The multiplication matrices S_k (k=0,1, ... ,7) are defined as follows (again we only list the nonzero entry values s_{ij}, where i,j=1-8).

$$S_0 = \{s_{ij} | s_{11} = 1, s_{22} = 0.75, s_{32} = 0.25, s_{33} = 0.5, s_{43} = 0.5, s_{44} = 0.25, s_{54} = 0.75, s_{65} = 1, s_{76} = 0.75, s_{86} = 0.25, s_{87} = 0.5\}$$
$$S_1 = \{s_{ij} | s_{17} = 0.5, s_{18} = 0.25, s_{28} = 0.75\}$$
$$S_2 = \{s_{ij} | s_{31} = 1, s_{42} = 0.75, s_{52} = 0.25, s_{53} = 0.5, s_{63} = 0.5, s_{64} = 0.25, s_{74} = 0.75, s_{85} = 1\}$$
$$S_3 = \{s_{ij} | s_{16} = 0.75, s_{26} = 0.25, s_{27} = 0.5, s_{37} = 0.5, s_{38} = 0.25, s_{48} = 0.75\}$$
$$S_4 = \{s_{ij} | s_{51} = 1, s_{62} = 0.75, s_{72} = 0.25, s_{73} = 0.5, s_{83} = 0.5, s_{84} = 0.25\}$$
$$S_5 = \{s_{ij} | s_{14} = 0.75, s_{25} = 1, s_{36} = 0.75, s_{46} = 0.25, s_{47} = 0.5, s_{57} = 0.5, s_{58} = 0.25, s_{68} = 0.75\}$$
$$S_6 = \{s_{ij} | s_{71} = 1, s_{82} = 0.75\}$$
$$S_7 = \{s_{ij} | s_{12} = 0.25, s_{13} = 0.5, s_{23} = 0.5, s_{24} = 0.25, s_{34} = 0.75, s_{45} = 1, s_{56} = 0.75, s_{66} = 0.25, s_{67} = 0.5, s_{77} = 0.5, s_{78} = 0.25, s_{88} = 0.75\}$$

The nonzero entry values in S_k's have the common factor of 0.25, which can be extracted and absorbed in the modified de-quantizer. Then the nonzero entry values in S_k become 4, 3, 2 and 1. We implement the down-sampling by 1.25 using a similar approach as that employed for the 1.50 case. Because of space constraint, the details of how this is done will not be given here. To get a block in the resulting image down-scaled by 1.25, the number of elementary operations needed is: $(6900 + 1680 + 7360 + 1792 + 10752)/16 = 1780$. On the other hand, the brute-force approach needs 2349 operations. The saving of the compressed domain approach is about 24%.

The DCT-domain upsampling operation can be derived in a similar manner by designing appropriate matrices S_i's. More details can be found in [13].

Fig. 2. Original image: Lena.

(a) (b)

Fig. 3. Lena image: (a) downsampled by a factor of 1.25 (b) downsampled by a factor of 1.50.

4. EXPERIMENTAL RESULTS

To evaluate the performance of the fractional down-sampling methods, we use the same set of test images from [5]. With an original image in spatial domain, we first transform it to DCT domain with 8×8-pixel blocks. Figure 2 shows the original Lena image

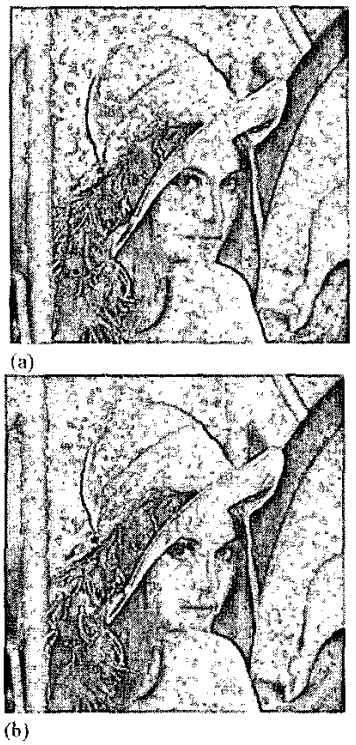

Fig. 4. Lena image: (a) reconstructed by downsampling and upsampling by a factor of 1.25; (b) reconstructed by downsampling and upsampling by a factor of 1.50.

and Figure 3 shows the Lena images obtained after filtering and downsampling by 1.25 and 1.50, respectively. It is clear that the downsampled images are sharp and have no visual artifacts.

We then reconstructed the image by upsampling after filtering and downsampling with our scheme. The reconstructed images with scaling factors of 1.25 and 1.50 for Lena are shown in Figure 4. We use PSNR (peak signal-to-noise ratio) to measure the objective quality of reconstructed images as compared to the original ones. The PSNR is defined as

$$PSNR = 10\log_{10}[255^2/(MeanSquareError)]$$

The PSNR values are tabulated in Table 1 and 2. For comparison, we also list the PSNR values obtained by the spatial-domain bilinear interpolation method. We can see that the PSNR values of our approach are much better than those of spatial-domain methods.

5. CONCLUSION

We have presented a scheme to scale image and video by fractional scaling factors of 1.50 and 1.25, which is in contrast to those algorithms that can only scale image and video by integer scaling factors directly in DCT domain. This helps to derive a wide range of scaling factors to support a variety of applications, such as the detection of variable sized face with a fixed-sized face model. We decompose the DCT computation processes into several common steps, and performs these steps sequentially in the vertical and horizontal directions. This gives rise to a simple and fast algorithm that can be easily implemented in both software and hardware. The resulting methods have been demonstrated to be computationally efficient and possess better PSNR measure as compared to the spatial domain bilinear interpolation method.

Table 1. PSNR values after downsampling and updampling using DCT-domain and spatial domain techniques (scaling by 1.50).

Image	Lena	Watch	F-16	Cap	Average
spatial domain	30.66	25.89	28.96	31.02	29.13
DCT domain	35.56	29.80	33.89	35.00	33.56

Table 2. PSNR values after downsampling and updampling using DCT-domain and spatial domain techniques (scaling by 1.25).

Image	Lena	Watch	F-16	Caps	Average
spatial domain	30.80	26.11	28.84	32.98	29.68
DCT domain	36.65	30.89	35.37	36.07	34.75

6. REFERENCES

[1] B. C. Smith and L. A. Rowe, "Algorithms for manipulation of compressed images," *IEEE Computer Graphics and Applications*, vol. 13, no. 5, pp. 34–42, Sept. 1993.

[2] A. Neri, G. Russo, and P.Talone, "Inter-block filtering and down-sampling in dct domain," *Signal Processing: Image Communication*, vol. 6, no. 4, pp. 303–317, Aug. 1994.

[3] S.-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of mc- dct compressed video," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 1, pp. 1–11, Jan. 1995.

[4] N. Merhav and V. Bhaskaran, "Fast algorithms for dct-domain down-sampling and inverse motion compensation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 7, no. 3, pp. 468–476, June 1997.

[5] R. Dugad and N. Ahuja, "A fast scheme for image size change in the compressed domain," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 11, no. 4, pp. 461–474, Apr. 2001.

[6] T.-S. Chua, Y. Zhao, and M.S. Kankanhalli, "Detection of human faces in a compressed domain for video stratification," *The Visual Computer*, vol. 18, no. 2, pp. 121–133, 2002.

[7] S. Acharya and B. Smith, "Compressed domain transcoding of mpeg," in *Proc. of IEEE Intl. Conf. Multimedia Computing and Systems*, 1998, pp. 295–304.

[8] Y. Arai, T. Agui, and M. Nakajima, "A fast dct-sq scheme for images," *The Trans. of the IEICE*, vol. E 71, no. 11, pp. 1095–1097, Nov. 1988.

[9] W. B. Pennebaker and J. Mitchell, *JPEG still image data compression standard*, Von Nostrand Reinhold, 1993.

[10] J. Song and B.-L. Yeo, "A fast algorithm for dct-domain inverse motion compensation based on shared information in a macroblock," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 10, no. 5, pp. 767–775, Aug. 2000.

[11] J. M. Adant et al., "Block operations in digital signal processing with application to TV coding," *Signal processing*, vol. 13, no. 4, pp. 385–397, 1987.

[12] B. Chitprasert and K. R. Rao, "Discrete cosine transform filtering," *Signal processing*, vol. 19, no. 3, pp. 233–245, Mar. 1990.

[13] Y. Zhao and M. Kankanhalli T.-S. Chua, "DCT-domain algorithms for fractional scaling and inverse motion compensation," Tech. Rep., School of Computing, National University of Singapore, 2001.