CHAPTER 6

COMPRESSED DOMAIN RESIZER

6.1. Introduction

The compressed domain resizer changes the resolution of compressed domain frame from higher resolution to lower resolution. This resizer works for any higher resolution input and any lower resolution output. Lower resolution shall be less than the input resolution, and shall be multiple of 16. The input to the compressed domain resizer comes from compressed domain decoder. The compressed domain decoded frame with higher resolution, is resized into smaller screen size, called as compressed domain resized frame here. The resizing operation is explained as follows.

$REZ = Cf \times v \times ((Cf^{-1} \times ORG \times Cf) \gg 32) \times h \times Cf^T$ .........................................................(6-1)

where, $ORG$ is the compressed domain decoded frame

$REZ$ is the resized frame

Here, $(Cf^{-1} \times ORG \times Cf) \gg 32$ is the inverse transform of compressed domain decoded frame that gives spatial domain decoded frame (spatial domain frame). $(v \times \text{spatial domain frame} \times h)$ is the process of applying horizontal and vertical filters on the spatial domain decoded frame to get spatial domain resized frame (spatial domain resized frame). The core forward transform is applied on spatial domain resized frame gives the compressed domain resized frame, i.e., compressed domain resized frame $= Cf \times \text{spatial domain resized frame} \times Cf^T$

This entire process is modified into sparse matrices as follows.

$REZ = A \times V \times (p \times l \times ORG \times l^T \times p^T) \times H \times A^T$ .................................................................(6-2)

The linear transform with double sided matrix multiplication processes are brought into different steps. The resizing process (which is shown in Fig. 6.1) is explained in the following steps.

1. Pre-processing: Conversion of compressed domain input frame to pseudo domain frame
2. Generation of indices and weights of filter
3. Horizontal resizing: Horizontal transformation of input frame to required output resized width
4. Vertical resizing: Vertical transformation of input frame to required output resized height
5. Post-processing: Conversion of pseudo domain resized frame to compressed domain frame

Input Frame \[\rightarrow\] Pre-processing

Resizing info - Width \[\rightarrow\] Generation of Horizontal Filter Indices and Weights \[\rightarrow\] Horizontal Filtering

Resizing info - Height \[\rightarrow\] Generation of Vertical Filter Indices and Weights \[\rightarrow\] Vertical Filtering \[\rightarrow\] Post-processing \[\rightarrow\] Resized Frame

**Fig. 6.1 Block diagram of compressed domain resizing**

### 6.2. Pre-processing

The compressed domain input frame is pre-processed so that the domain is shifted to pseudo domain. The pre-processing involved the fixed length transformation from compressed domain. It did not transform to exact spatial domain, because of its fixed length approximation of transformation. The pre-processing is applied on each 4x4 block of the compressed domain. This pre-processing is defined as follows.

\[PRE_{4x4} = p \times l \times ORG_{4x4} \times \bar{l}^T \times p^T\] ......................................................... (6-3)

where,

\[PRE_{4x4}\] is the preprocessed 4x4 block and \[ORG_{4x4}\] is the compressed domain 4x4 original block

\[p = \begin{pmatrix} 2 & 2 & 2 & 1 \\ 2 & 1 & -2 & -2 \\ 2 & -1 & -2 & 2 \\ 2 & -2 & 2 & -1 \end{pmatrix}\] and \[l = \begin{pmatrix} 0.125 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.125 & 0 \\ 0 & 0 & 0 & 0.1 \end{pmatrix}\]
\[ p \text{ and } l \text{ are derived from } Cf^{-1} = \begin{pmatrix} 0.25 & 0.2 & 0.25 & 0.1 \\ 0.25 & 0.1 & -0.25 & -0.2 \\ 0.25 & -0.1 & -0.25 & 0.2 \\ 0.25 & -0.2 & 0.25 & -0.2 \end{pmatrix} \]

Neri, M. et al. (1997) approached the matrix multiplication by decimating into simple steps. In the same way, applying the ‘l’ and ‘l^T’ coefficients on the ORG_{4x4} matrix is carried out to get \( x_2 = l \times ORG_{4x4} \times l^T \) to be compatible with Hardware. In order to have finite length for coefficients, the coefficients of \( l \) has been changed to:

\[
l = \begin{pmatrix} 0.125 & 0 & 0 & 0 \\ 0 & 0.125 & 0 & 0 \\ 0 & 0 & 0.125 & 0 \\ 0 & 0 & 0 & 0.125 \end{pmatrix}.
\]

The result is not equal to true spatial domain block. But it reflects the pattern of spatial domain block. So this domain is called 'pseudo domain'. As this reflects the pattern, this pseudo domain blocks can be used for spatial domain motion search or spatial domain processing. Applying the ‘p’ and ‘p^T’ coefficients on the \( x_2 \) matrix is done in the following way to get RES = \( p \times x_2 \times p^T \). Here also, this matrix multiplication is simplified to suit in hardware which performs horizontally and vertically.

### 6.3. Generation of weights and indices of filter

Erik, M. et al. (2003) explained the third and fourth order cubic convolution interpolation which is adopted as base function for resizing in this research work. This method is applicable for horizontal and vertical filtering. The length of output and input in either horizontally or vertically decides the term \( \text{ratio} \). This refers the resizing ratio in 1D.

\[
\text{ratio} = \frac{ol}{il} \hspace{1cm} ................................................................. (6-4)
\]

where, \( ol \) is the output length and \( il \) is the input length.

The \textit{index} for each position \((p)\) in output line is calculated as follows.

\[
\text{index}(p) = \left[ \text{centre}(p) - \frac{\text{kernelwidth}}{2} + 1 \right], \quad p = 1 \ldots ol \hspace{1cm} .................................................. (6-5)
\]

where

\[
\text{kernelwidth} = 12 \text{ is used as default to address higher resizing ratio.}
\]

\[
\text{centre}(p) = \frac{p - 0.5}{\text{ratio}} + 0.5 \hspace{1cm} ..................................................(6-6)
\]

The deviation of indices with respect to the centre of filter is calculated as follows for every \( p \).
The magnitudes of filter are calculated as follows.

\[ \text{magnitude}(p, f) = \text{ratio} \times \varphi_{cc}(\text{ratio} \times (\text{dev}(p) - f)), \quad f = 1 \ldots \text{kernelwidth} \]  \hspace{1cm} (6-8) 

The kernels are cubic, nearest, linear, lanczos2 and lanczos3 given by Erik, M. et al. (2003).

Cubic Kernel is \( \varphi_{cc}(x) = \) 
\[
\begin{cases} 
  \frac{3}{2} |x|^3 - \frac{5}{2} |x|^2 + 1, & 0 < |x| \leq 1 \\
  -\frac{1}{2} |x|^3 + \frac{5}{2} |x|^2 - 4|x| + 2, & 1 < |x| \leq 2 \\
  0, & \text{else}
\end{cases}
\]  \hspace{1cm} (6-9) 

Nearest Kernel is \( \varphi_{cc}(x) = \) 
\[
\begin{cases} 
  1, & -0.5 \leq x < 0.5 \\
  0, & \text{else}
\end{cases}
\]  \hspace{1cm} (6-10) 

Linear Kernel is \( \varphi_{cc}(x) = \) 
\[
\begin{cases} 
  x + 1, & -1 \leq x < 0 \\
  1 - x, & 0 \leq x \leq 1
\end{cases}
\]  \hspace{1cm} (6-11) 

Lanczos2 Kernel is \( \varphi_{cc}(x) = \) 
\[
\begin{cases} 
  \frac{\sin(\pi x) \times \sin(\frac{\pi x}{2} + \epsilon)}{\frac{\pi x}{2}}, & |x| < 2 \\
  \frac{\sin(\pi x) \times \sin(\frac{\pi x}{3} + \epsilon)}{\frac{\pi x}{3}}, & |x| < 3 \\
  0, & \text{else}
\end{cases}
\]  \hspace{1cm} (6-12, 6-13) 

Lanczos3 Kernel is \( \varphi_{cc}(x) = \) 
\[
\begin{cases} 
  \frac{\sin(\pi x) \times \sin(\frac{\pi x}{2} + \epsilon)}{\frac{\pi x}{2}}, & |x| < 2 \\
  \frac{\sin(\pi x) \times \sin(\frac{\pi x}{3} + \epsilon)}{\frac{\pi x}{3}}, & |x| < 3 \\
  0, & \text{else}
\end{cases}
\]  \hspace{1cm} (6-12, 6-13) 

Out of all kernels, cubic kernel is used here for analysis.

The weights of filter are calculated as follows.

\[ \text{weight}(p, f) = \frac{\text{magnitude}(p, f)}{\sum_{f=1}^{\text{kernelwidth}} \text{magnitude}(p, f)}, \quad p = 1 \ldots \text{ol} \]  \hspace{1cm} (6-14) 

In the case of indices being negative, the weights fall on those negative indices along the kernel width are added to have weight at \( \text{index} = 0 \). In the case of indices with kernel width being greater than the input length \((il)\), the weights fall on those exceeding indices along the kernel width are added to have weight at \( \text{index} = \text{ol} - 1 \). All the weights are shifted left nine bits and truncated. More than 9-bits accuracy, the quality of video improves. But if the bit accuracy is less than 9-bits, the subjective quality is bad. So the weights will have 9-bits accuracy which is enough for good video quality (>50dB compared with \text{imresize.m} of MATLAB function).

So for horizontal filter, \( H = [h \times 2^9] \) and for vertical filter, \( V = [v \times 2^9] \)……(6-15)
This is done as follows.

\[
\begin{align*}
    l_1 &= \text{weights} \times 2 \\
    l_2 &= \lfloor l_1 \times 4 \rfloor \\
    l_3 &= \lfloor l_2 \times 8 \rfloor \\
    l_4 &= \lfloor l_3 \times 16 \rfloor \\
    l_5 &= \lfloor l_4 \times 32 \rfloor \\
    l_6 &= \lfloor l_5 \times 64 \rfloor \\
    l_7 &= \lfloor l_6 \times 128 \rfloor \\
    l_8 &= \lfloor l_7 \times 256 \rfloor \\
    l_9 &= \lfloor l_8 \times 512 \rfloor
\end{align*}
\]

weights = \(l_1 \ll 9 + l_2 \ll 7 + l_3 \ll 6 + l_4 \ll 5 + l_5 \ll 4 + l_6 \ll 3 + l_7 \ll 2 + l_8 \ll 1 + l_9\) .................. \(6-16\)

Now the weights are represented as integers. These weights are directly used for filtering.

6.4. Horizontal Filtering

Each line in the input frame is resized to required width. The term ‘index’ is the horizontal co-ordinate in a line of frame. A set of filter coefficients (integer) are applied to the original frame starting from index to the filter width (kernel width) as shown in Fig. 6.2. This process is represented by Equation (6-17).

\[\text{Hor Resized frame} = \text{PRE} \times H\] ................................................................. \(6-17\)

Each location in a horizontal line of resized frame will have its index, set of filter coefficients called weights, that have to be applied on preprocessed frame. Those indices and filter coefficients are derived by bi-cubic convolution interpolation. The bi-cubic convolution interpolation technique (one of the Polyphase-filtering techniques) is adopted to generate the filter coefficients (weights) and shift position values (index) of the horizontal filter operations. The indices and weights are denoted by indices and hweights in horizontal filtering.

![Fig. 6.2 The Horizontal Filtering on a single line by filter (with Kernel Width = 7)](image-url)
After horizontal filtering, all the values in the resized frame are scaled down keeping one bit precision as follows, because the horizontal filter weights are already scaled by $2^9$.

\[
\text{Hor Resized frame} = \left[ \frac{\text{Hor Resized frame}}{2^9} \right]
\]  
(6-18)

The horizontal filter resizes the frame horizontally (width reduction, keeping height same). The process is applied to Chroma components also. But the \( \text{hindices} \) and \( \text{hweights} \) for chroma components are denoted as \( \text{chindices} \) and \( \text{chweights} \).

### 6.5. Vertical Filtering

Each column in the horizontally resized frame is resized to required height. The term 'index' is the vertical co-ordinate in a column of frame. A set of filter coefficients (integer) are applied to the horizontally resized frame starting from index to the filter width (kernel width) as shown in Fig. 6.3.

\[
\text{Ver Resized frame} = V \times \text{Hor Resized frame}
\]  
(6-19)

Each location in a vertical line of resized frame will have its index, set of filter coefficients called \( \text{weights} \). The \( \text{indices} \) and \( \text{weights} \) are denoted by \( \text{vindices} \) and \( \text{vweights} \) in vertical filtering.

![Fig. 6.3 The Vertical Filtering on a single column by filter (with Kernel Width = 6)](image-url)
After vertical filtering, all the values in the resized frame are scaled down as follows.

\[ rez = \left[ \frac{Ver \ Resized \ frame}{2^{10}} \right] \]  

(6-20)

The vertical filter resizes the frame vertically (height reduction, keeping width same). The process is applied to Chroma components also. But the \textit{vindices} and \textit{vweights} for Chroma components, which are calculated, are denoted as \textit{cvindices} and \textit{cvweights}. The vertical filter resizes the horizontally reduced frame vertically.

\section*{6.6. Post-processing}

Post-processing is the method converting the resized frame from its pseudo domain values to compressed domain values. This is done by the following equation on a 4x4 block basis.

\[ REZ_{4x4} = A \times rez_{4x4} \times A^T \]  

(6-21)

where,

\[ A = \begin{bmatrix} 
1 & 1 & 1 & 1 \\
2 & 1 & -1 & -2 \\
1 & -1 & -1 & 1 \\
1 & -2 & 2 & -1 \\
\end{bmatrix} \]

The resizing is concluded with post processing.

\section*{6.7. Conclusion of Compressed Domain Resizer}

The purpose of compressed domain resizer is to resize the screen size of compressed domain to smaller size from bigger size. The process involved only integer operations as the weights are modified to integers. The application of filters on the frame is done layer by layer where each layer has either 0 or ±1 to be compatible with Hardware specifications. The compressed domain resizer was developed and coded in MATLAB. The compressed domain decoded frame is resized by compressed domain resizer. The results are converted to spatial domain frame and kept separately. Then the input of the compressed domain resizer is converted to spatial domain frame. This spatial domain frame is resized by \texttt{imresize.m} of MATLAB function, to get resized spatial domain frame. This spatial domain frame (which \texttt{imresize.m} modified) is compared with the spatial domain frame of resizer. This experiment resulted, for all the frames of all sequences, the PSNR was found more than 50dB, which is better visual quality.