Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Efficient implementation of a cubic-convolution based image scaling engine*

Xiang WANG†, Yong DING†‡, Ming-yu LIU, Xiao-lang YAN

(*Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China*)

†E-mail: {wangxiang, dingy}@vlsi.zju.edu.cn

Received Feb. 18, 2011; Revision accepted June 30, 2011; Crosschecked Aug. 22, 2011

Abstract: In video applications, real-time image scaling techniques are often required. In this paper, an efficient implementation of a scaling engine based on 4×4 cubic convolution is proposed. The cubic convolution has a better performance than other traditional interpolation kernels and can also be realized on hardware. The engine is designed to perform arbitrary scaling ratios with an image resolution smaller than 2560×1920 pixels and can scale up or down, in horizontal or vertical direction. It is composed of four functional units and five line buffers, which makes it more competitive than conventional architectures. A strict fixed-point strategy is applied to minimize the quantization errors of hardware realization. Experimental results show that the engine provides a better image quality and a comparatively lower hardware cost than reference implementations.

Key words: Cubic-convolution, Hardware implementation, Interpolation, Engine
doi:10.1631/jzus.C1100040 **Document code:** A **CLC number:** TN79+1; TP752

1 Introduction

As various kinds of digital display devices have become popularized, several types of image formats have been utilized. The resolution of a digital display, for example, LCD, is fixed according to product specifications. It is necessary to convert various resolutions of input images to adapt them to a display device. Thus, image interpolation becomes an important image processing operation applied in such diverse fields as consumer electronics, medical imaging, and military applications. When the image is interpolated from a lower resolution to a higher resolution, it is traditionally called image scaling up or up-scaling. Similarly, image scaling down or down-scaling means interpolation from a higher resolution to a lower resolution. Images are usually provided at a fixed size and need to be scaled either up or down for a variety of uses as mentioned above.

Interpolation functions can be used to generate interpolated images (Hou and Andrews, 1978; Keys, 1981; Lehmann *et al.*, 1999; Shi and Reichenbach, 2006). Most published interpolation methods have been developed by interpolating the pixels based on the characteristics of local features, such as edge information (Hong *et al.*, 1996) or neighboring pixel information. Therefore, according to these differences, interpolation methods can be classified into two categories: adaptive and non-adaptive methods. Adaptive interpolation algorithms (Li and Orchard, 2001; Shi and Ward, 2002; Arandiga *et al.*, 2003; Chen *et al.*, 2009) rely on the image features of the source image, and the computational logic is also dependent on the image features. As a result, these methods need more complex computational logic and heavy loading to a real-time system that is usually realized on software. The non-adaptive algorithms do not rely on the image features, and the same computational logic is repeated in every pixel or group of local pixels irrespective of the image features. Thus, certain computations are performed indiscriminately

‡ Corresponding author

* Project supported by the National High-Tech R & D Program (863) of China (No. 2009AA011706) and the Fundamental Research Funds for the Central Universities (No. KYJD09012)

to the whole image. The non-adaptive algorithms are easy to implement on hardware, especially on real-time display devices.

The complexity and performance of scaling engines vary depending on the interpolation algorithm used. Several non-adaptive interpolation algorithms have been compared (Parker *et al.*, 1983; Lehmann *et al.*, 1999). The simplest and fastest algorithm is nearest neighbor (NN) interpolation, but it incurs the largest error. Bilinear interpolation has been applied on some hardware implementations (Kim *et al.*, 2003; Aho *et al.*, 2007). The bilinear algorithm is simple enough that it needs only four (2×2) neighboring points for interpolating. Although the quality of the bilinear algorithm is much better than that of NN interpolation, it still has the undesirable features of block and blur effects. Winscale (Kim *et al.*, 2003) is a modified linear scaling algorithm which scales using an area pixel model. Hou and Andrews (1978) proposed the cubic spline method for image scaling. These interpolation kernels are based on the basis spline (B-spline) function. The B-spline interpolation has an excellent performance for image scaling, but the coefficients for B-spline interpolation kernels are infinite. Thus, it is difficult to implement on hardware. Hou and Andrews (1978) also developed the cubic spline approximation. This approximation kernel is easy to realize on hardware but causes strong blur effects on the image (Her and Yuan, 1994). Keys (1981) developed a 4×4 cubic convolution interpolation algorithm, which is a little more complex than the NN and bilinear interpolations, but much simpler than B-spline interpolation. Clearly, this higher order model may provide a better quality of interpolation. Furthermore, it is feasible to be realized on hardware (Nuno-Maganda and Arias-Estrada, 2006; Lin *et al*., 2008). Thus, cubic convolution provides a relatively accurate interpolation, as required for the modern display devices (Parker *et al.*, 1983).

Several non-adaptive interpolation algorithms are reported to have been implemented in very-large-scale integration (VLSI) design. Aho *et al.* (2007) proposed an implementation for dealing with large resolution color videos in real-time. The design is based on bilinear interpolation, and is only able to scale down the image. The VLSI realization of Winscale (Kim *et al.*, 2003) has a better performance than bilinear interpolation. The computational complexity of the algorithm is as low as that of bilinear interpolation but it needs four line buffers for vertical scaling, which is much more than that required for conventional bilinear design, thus making it more expensive. The cubic and bisigmoidal interpolation method (Feng *et al.*, 2001) uses cubic convolution interpolation in horizontal scaling and bisigmoidal interpolation in vertical scaling. It has been shown to give slightly better quality than bilinear interpolation with scaling up ratios restricted to 4/5, 9/10, and 24/25. Lin *et al.* (2008; 2010) proposed a novel implementation, called first-order polynomials convolution, in which both cubic convolution interpolation and its improved algorithm are realized. Although this structure has the best performance of all of these VLSI designs, the realization is too complex. It uses too many multipliers, which results in a huge hardware cost.

This paper presents an efficient real-time implementation of a scaling engine with a 4×4 cubic convolution interpolation kernel. The engine is designed to process source and destination images that are smaller than 2560×1920 pixels. It can perform scaling up or down for arbitrary scaling ratios in horizontal and vertical directions. It is composed of only two interpolation filters, two control modules, and five line buffers. Moreover, the truncated error between the hardware implementation and software floating simulation is no larger than one for each pixel. There is also no cumulative error, as it processes the image pixel by pixel.

2 Interpolation filter

2.1 Cubic convolution interpolation

The proposed cubic interpolation, a 2D interpolation method, uses 4×4 neighboring pixels for interpolating. It is usually composed of two 1D interpolations, a horizontal and a vertical interpolation, to reduce the complexity. Eq. (1) shows the 1D interpolation kernel of four-point cubic convolution:

$$h(x)=\begin{cases}\dfrac{3}{2}|x|^3-\dfrac{5}{2}|x|^2+1, & 0<|x|<1,\\ -\dfrac{1}{2}|x|^3+\dfrac{5}{2}|x|^2-4|x|+2, & 1<|x|<2,\\ 0, & \text{otherwise.}\end{cases} \quad (1)$$

Fig. 1 illustrates the 2D interpolation process in which pixels are abstracted into points located in the center of grids. The destination image (Fig. 1b) is interpolated from the source image (Fig. 1a), so the border pixels of the two images must coincide with each other, as shown in Fig. 1c.

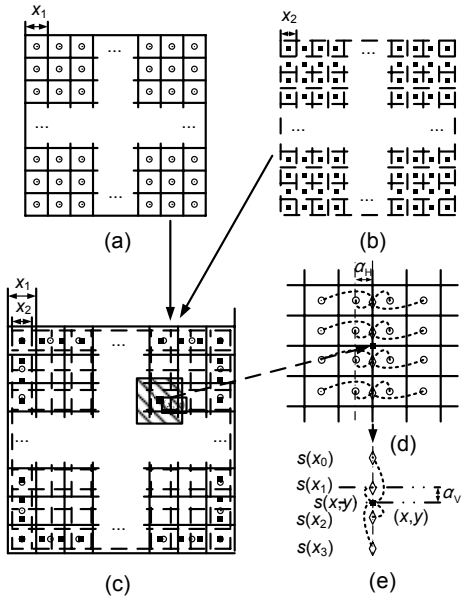

Fig. 1 Illumination of 2D interpolation at point (x,y)
(a) Grid of the source image; (b) Grid of the destination image, whose resolution is different from that of the source image, where x_1 and x_2 are the grid widths of the source and destination pixels, respectively; (c) Comparison between source and destination images; (d) Snapshot of (c), where α_H is the horizontal offset between the second column source pixels and intermediate results; (e) 1D interpolation in the vertical direction, where α_V is the vertical offset between the second source pixel $s(x_1)$ and the destination pixel $s(x,y)$

To obtain the destination pixel of $s(x,y)$, two steps of interpolation are taken. Firstly, 1D interpolation is executed in the horizontal direction with four hollow diamond points (Fig. 1d), and the intermediate results denoted as $s(x_0)$, $s(x_1)$, $s(x_2)$, and $s(x_3)$ are calculated. Then they are interpolated in the vertical direction to obtain the final destination pixel signal $s(x,y)$ (Fig. 1e). Note that the interpolating sequence between the horizontal direction and vertical direction has no effect on the final result. Thus, the order of interpolation in the two directions can be adjusted to meet the requirements of the architecture.

2.2 Cubic convolution interpolation

The processes of interpolation in both the horizontal and vertical directions are similar. Here, we discuss only the structure of the 1D interpolation filter. There are four source pixels $f(x_0)$, $f(x_1)$, $f(x_2)$, and $f(x_3)$, and a destination pixel $s(x)$ of 1D interpolation. As the space between two neighboring sampled points is 1, according to Eq. (1), the interpolation formula is derived by

$$\begin{aligned} s(x) = {} & f(x_0)\left[-\frac{1}{2}(\alpha+1)^3+\frac{5}{2}(\alpha+1)^2-4(\alpha+1)+2\right] \\ & + f(x_1)\left(\frac{3}{2}\alpha^3-\frac{5}{2}\alpha^2+1\right) \\ & + f(x_2)\left[\frac{3}{2}(1-\alpha)^3-\frac{5}{2}(1-\alpha)^2+1\right] \\ & + f(x_3)\left[-\frac{1}{2}(2-\alpha)^3+\frac{5}{2}(2-\alpha)^2-4(2-\alpha)+2\right], \end{aligned} \tag{2}$$

where α is the offset between the second source pixel $f(x_1)$ and the destination pixel $s(x)$.

In our proposed architecture, the Farrow structure is employed, which is very useful for a polynomial based interpolation filter (Farrow, 1988; Erup *et al.*, 1993; Gardner, 1993). It is very well suited for high-speed real-time implementation because it uses a small number of multiplications and greatly reduces the complexity of hardware implementation.

To fit our Farrow based structure of the interpolation filter, Eq. (2) can be rewritten as

$$\begin{aligned} s(x) = {} & \left\{\left[\left(-\frac{1}{2}f(x_0)+\frac{3}{2}f(x_1)-\frac{3}{2}f(x_2)+\frac{1}{2}f(x_3)\right)\alpha\right.\right. \\ & \left.+\left(f(x_0)-\frac{5}{2}f(x_1)+2f(x_2)-\frac{1}{2}f(x_3)\right)\right]\alpha \\ & \left.+\left(-\frac{1}{2}f(x_0)+\frac{1}{2}f(x_2)\right)\right\}\alpha+f(x_1). \end{aligned} \tag{3}$$

The Farrow coefficients can be derived from Eq. (3). Our proposed Farrow based interpolation filter consists of four columns of finite impulse response (FIR) transversal filters, and each FIR column has four taps. The tap coefficients are listed in Table 1.

The coefficients of the filter are all integrals of 1/2, so they can be performed by plus, left shift, and right shift operations. Fig. 2 shows the filter implementation structure. The inputs are the four sampled data points $f(x_0)$, $f(x_1)$, $f(x_2)$, $f(x_3)$, and the offset α, and

the output is the interpolated result $s(x)$. With three multiplications, five shifters and ten adders are required in the pipelining of the structure. The interpolation filter needs seven clocks to accomplish the function.

Table 1 Coefficients of Farrow based interpolation filter

i	l=0	l=1	l=2	l=3
0	0	−1/2	1	−1/2
1	1	0	−5/2	3/2
2	0	1/2	2	−3/2
3	0	0	−1/2	1/2

i: tap number; l: column number

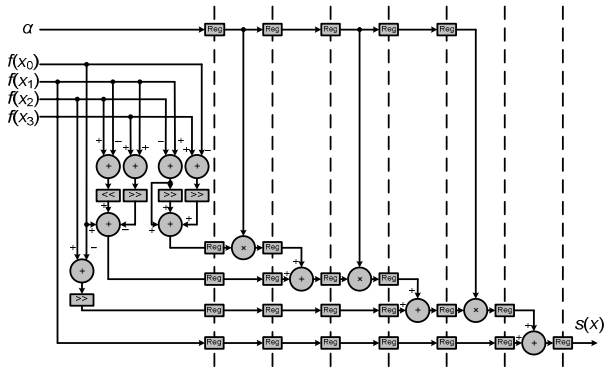

Fig. 2 Structure of the interpolation filter

3 Fixed-point strategy on hardware

3.1 Fixed-point strategy of offset

The Farrow structure needs to transfer only the offset α for each interpolation. Thus, to improve the precision of our proposed scaling engine, a strict fixed-point of offset α should be used.

The offset is derived by accumulating the horizontal scaling factor α_H or vertical scaling factor α_V. The scaling factors can be derived as follows:

$$\alpha_H = \frac{H_{Src}-1}{H_{Des}-1}, \tag{4}$$

$$\alpha_V = \frac{V_{Src}-1}{V_{Des}-1}, \tag{5}$$

where H_{Src} and V_{Src} are the horizontal and vertical resolutions of the source image, respectively, and H_{Des} and V_{Des} are the horizontal and vertical resolutions of the destination image, respectively.

The proposed engine is designed to process images smaller than 2560×1920 pixels, so two 33 bits registers are used to store the horizontal and vertical offsets.

Three variables are stored in the horizontal offset register (Fig. 3). The relationship between these variables and the horizontal scaling factor α_H is given as

$$\frac{H_{Src}-1}{H_{Des}-1}\times 2^9 = \text{Integer}\times 2^9 + \text{H\_step} + \text{H\_comp}/\text{H\_des}, \tag{6}$$

where Integer represents the integral result of the division $(H_{Src}-1)/(H_{Des}-1)$.

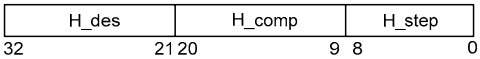

Fig. 3 Horizontal offset register read/write (R/W)

The divisor $H_{Des}-1$ is stored in the first 12 bits ([32:21]) of the register named H_des, and the truncated value for calculating the offset α is kept in the neighboring 12 bits ([20:9]) of the register named H_comp. The value of offset α is stored in the last 9 bits ([8:0]) of the register named H_step.

Initially, H_comp and H_step are set to 0. When the accumulator works, H_comp and H_step will be calculated in turn, and the results will be written in the corresponding positions. The integer part and the offset α are exported as the outputs of the accumulator.

In our proposed method, the truncated error of offset α is $O(2^{-9})$. The truncate value of the offset, which is smaller than 2^{-9}, is kept in H_comp when the scaling process is running, so there is no cumulative error.

To evaluate the performance of our proposed fixed-point strategy of offset α, we used the LIVE image quality assessment database (Wang *et al.*, 2004; Sheikh *et al.*, 2006; 2010). The images were interpolated with 6, 7, 8, 9, 10, and 11 bits of offset α. We also give the floating calculating results as reference. To obtain the peak signal-to-noise ratio (PSNR) in Table 2, these images were scaled in four different ways with the same scale factor, 4/3. The orders of the four different ways were as follows: scaling up in both the horizontal (H up) and vertical directions (V up); scaling up in the horizontal direction and scaling down in the vertical direction (V

Table 2 Peak signal-to-noise ratio (PSNR) of offset under different fixed-point methods

Scaling way	PSNR (dB)						
	6 bits	7 bits	8 bits	9 bits	10 bits	11 bits	Floating
H up & V up	45.79	46.08	46.19	46.22	46.23	46.24	46.24
H up & V down	39.55	39.66	39.69	39.70	39.71	39.71	39.71
H down & V up	38.95	39.04	39.07	39.09	39.09	39.09	39.09
H down & V down	36.82	36.89	36.91	36.92	36.92	36.92	36.92

down); scaling down in the horizontal direction (H down) and scaling up in the vertical direction; scaling down in both the horizontal and vertical directions. To determine the accuracy of the experiment, we chose 4/3 as the scaling factor since it is indivisible by 2^n. We then rescaled these images back to their original size to compare with the reference.

Table 2 shows PSNR comparisons of luminance of the offset α under different fixed-point strategies. Simulations show that with the increase in bits of the offset α, the processing results are more accurate. However, when the bits of the offset α are greater than 9, the improvement in PSNR is negligible. Considering the tradeoff between the cost of registers and performance, we chose 9 bits data to represent the offset α.

3.2 Fixed-point process of interpolation filter

The luminance component and two channels of chrominance components all consist of 8 bits data (0–255) whereas the offset α is 9 bits data. Thus, if all the fractional bits of the intermediate calculation data are kept, 36 bits data will be derived before exporting. Although keeping the fractional bits to the end of the filter and rounding them together (global rounding) are more accurate, it needs much wider registers to retain the unimportant information and a larger area to implement the filter. Thus, there should be a fixed-point process for the intermediate calculation of the interpolation filter. Considering the tradeoff between the computational accuracy and the design cost, we propose to keep only 4 bits of fraction when the intermediate data are written into the flip-flops during each cycle of calculation and to finally round the result before exporting it. Under the premise of accuracy, shorter registers can be taken to store the intermediate data in the calculations.

To evaluate the performance of our proposed fixed-point method of the interpolation filter, the test images in the database were processed with different fixed-point methods including keeping 0, 2, 4, 6, and 8 bits fractions of the intermediate data in the interpolation filter. To obtain the reference, we also ran a group of simulations which calculates with the floating method and uses global rounding before the output of the filter. To obtain the performance of a single interpolation filter, these images were scaled up or down in only one direction with the factor 4/3. In this way, the mean square error (MSE) of a single interpolator could be estimated.

Fig. 4 shows the MSE values obtained from comparisons of the test images. Note that the pixel errors of all the images do not exceed 1, which is attributed to the accurate fixed-point strategy of offset α. Compared with the global rounding method, if we adopt a no-fraction method (0 bit fraction shown in Fig. 4) or a 2 bits fraction method, the MSE is so large that nearly 20% of pixels of the processed image are error pixels, which may badly affect the quality of image. If the fixed-point strategy of keeping 4 bits to store the fraction is chosen, the percentage of error pixels in the images falls sharply to below 5%. Therefore, retaining 4 bits fractions of the intermediate data in the interpolation filter gives a better tradeoff between the image quality and the hardware cost.

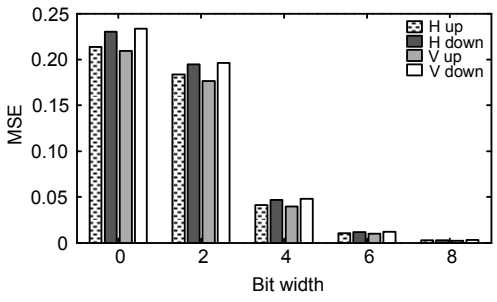

Fig. 4 Mean square error (MSE) under different fixed-point methods with different sizes

4 Architecture of scaling engine

In total, there are four logic units (including two combinational units and two control units) and five line buffers in the scaling engine. The four logic units are a source clock domain data read/write controller (Src_Ctrl), a destination clock domain data read controller (Des_Ctrl), a horizontal interpolator (H_Intp), and a vertical interpolator (V_Intp). The line buffers are dual-port static random access memory (SRAM).

The inputs and outputs of the scaling engine work at different operating frequencies. The scaling engine needs to deal with the problem of the data that cross clock domains. Thus, all the functional units are divided into two parts, one part working in the source clock cycle (T_{Src}), and the other part working in the destination clock (T_{Des}). T_{Des} is determined by T_{Src} and the ratio of source resolution to destination resolution, given by

$$T_{\mathrm{Des}} = T_{\mathrm{Src}} \cdot \frac{H_{\mathrm{Src}} V_{\mathrm{Src}}}{H_{\mathrm{Des}} V_{\mathrm{Des}}}. \tag{7}$$

To realize the arbitrary scaling ratios up/down in the horizontal and vertical directions, four different cases need to be discussed (Fig. 5).

Generally, the number of destination pixels generated by the scaling down interpolation is smaller than the number of source pixels. The scaling down interpolation takes a few clock cycles to produce a destination pixel, so it can work in the source clock domain. In contrast, the number of destination pixels generated by the scaling up interpolation is greater than the number of source pixels. As it is impossible to output more than one destination pixel in one source clock cycle, the scaling up interpolation must be arranged in the destination clock cycle. In the different cases as shown in Fig. 5, H_Intp and V_Intp are arranged in different clock domains. The principle is: to execute scaling down interpolation in the source clock domain, to cross the clock domain before executing the scaling up interpolation, and to arrange the corresponding interpolation filter to work in the destination clock.

To control the read and write of the line buffers more flexibly, dual-port SRAM is used instead of first in first out (FIFO). The write control signal of the line buffer is provided by Src_Ctrl and the read

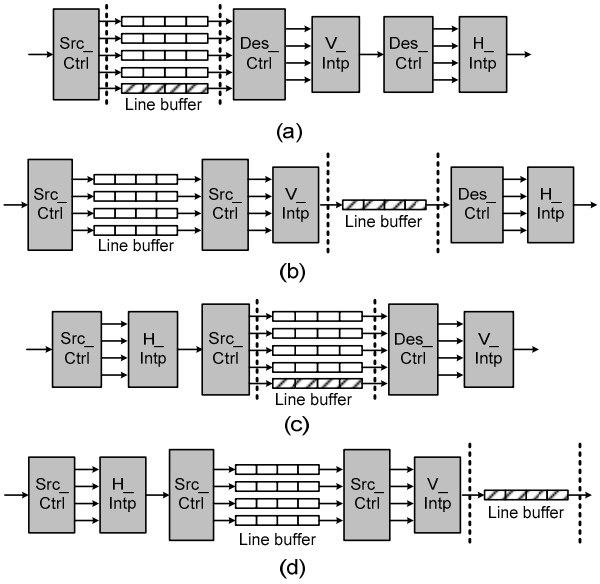

Fig. 5 Data flow of the scaling engine

(a) Horizontal up and vertical up; (b) Horizontal up and vertical down; (c) Horizontal down and vertical up; (d) Horizontal down and vertical down

control signal by Src_Ctrl or Des_Ctrl, according to the case. If the line buffer offers data to the interpolator in the source clock domain, the read control signal is provided by Src_Ctrl. Otherwise, it is provided by Des_Ctrl.

In the case of horizontal scaling up and vertical scaling up (Fig. 5a), both H_Intp and V_Intp work in the destination clock domain. The horizontal offset and vertical offset are provided by Des_Ctrl. The five line buffers cross the clock domain together. Src_Ctrl sends a control signal to write source image pixel data into the line buffer. When the line buffers have stored enough data, Des_Ctrl sends a control signal to read the data from line buffers for interpolating. The vertical interpolation executes first, and data from the line buffers are read in parallel.

Fig. 6 shows the data loading for V_Intp. The four vertical neighboring pixels are sent to V_Intp in the order shown on the left side of Fig. 6. At the same time, the fifth line buffer (shaded in Fig. 6) receives the source pixel data. When a line of interpolation has been finished, the integer part of the accumulating result of the vertical offset register is used as a switch. If the output is 1, it means that the vertical position of the next destination pixel crosses a source line from the current line. Thus, the data loading changes from Lines A, B, C, and D to Lines B, C, D, and E. The

preparing line buffer (shaded) also changes from Line E to Line A. If the integer output of the accumulation is still 0, it means that the next destination pixel is still located between the current source lines. Thus, Des_Ctrl reloads the data from the same four line buffers, that is Lines A, B, C, and D.

Note that no line buffer is used between the two interpolation filters in Fig. 5a, because they are both in the destination clock domain. The output of V_Intp enters Des_Ctrl, in which four D-flip-flops are employed as the time delay generators to buffer the data (Fig. 7). The four horizontal neighboring pixel data are then sent to H_Intp in parallel.

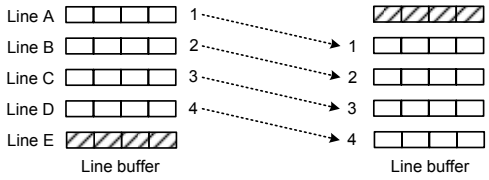

Fig. 6 Data loading for V_Intp

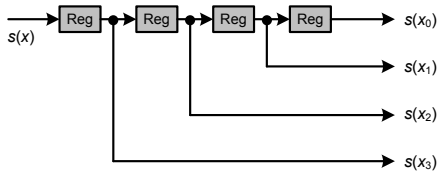

Fig. 7 Data loading for H_Intp

In Fig. 5b, the vertical direction is scaling down, so V_Intp works in the source clock domain and the vertical offset is provided by Src_Ctrl. Four line buffers are arranged in the source clock domain. The read and write of these four line buffers are in the same clock domain. The fifth line buffer (shaded in Fig. 5b) crosses the clock domain. H_Intp works in the destination clock domain. Des_Ctrl reads data from the fifth line buffer and uses the four-D-flip-flop block to send the data to H_Intp. It also provides the horizontal offset to H_Intp.

The crossing clock domain pattern of the case in Fig. 5c is similar to the case in Fig. 5a. Because it is scaling down in the horizontal direction, the engine executes the horizontal interpolation first, and then uses the five line buffers to cross the clock domain and executes the vertical interpolation.

The case in Fig. 5d is scaling down in both directions, so the two interpolation filters and four line buffers work in the source clock domain. After the vertical and horizontal interpolations finish, Des_Ctrl controls the read of data from the fifth line buffer (shaded) and finally exports them as the output of the engine in the destination clock domain. In this case, there are no data passing through Des_Ctrl, and Des_Ctrl provides only the control signal, so it is not shown in Fig. 5.

The architecture of the scaling engine is shown in Fig. 8. As well as the units mentioned above, a few

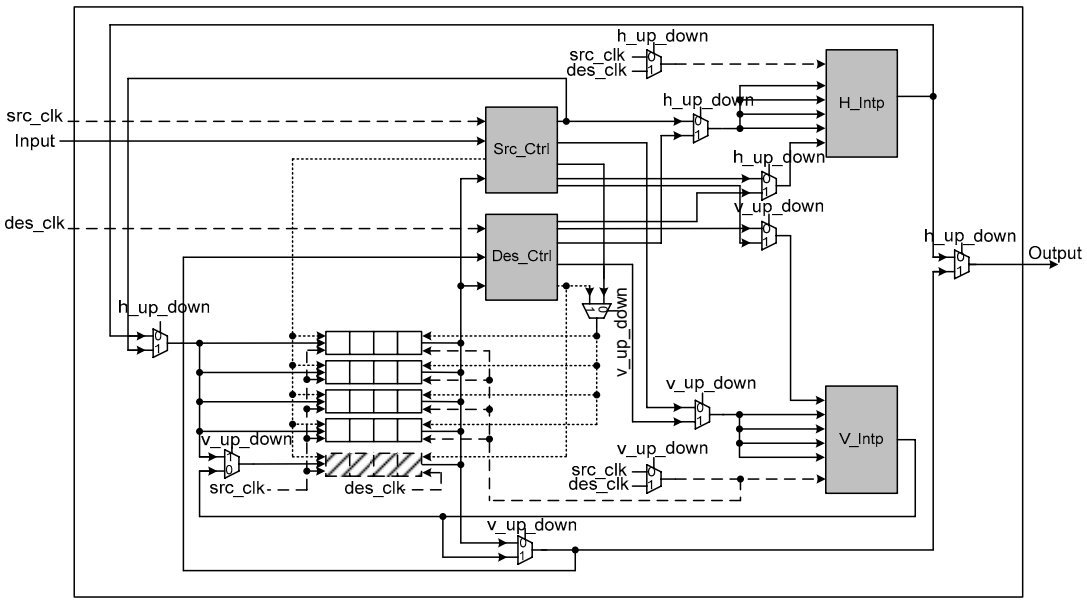

Fig. 8 Architecture of the scaling engine

multiplexers (MUXs) are employed. These MUXs are controlled by the signals h_up_down or v_up_down. If an image is scaled up in the horizontal direction, h_up_down is 0, and vice versa. Likewise for v_up_down. These MUXs control the engine so as to work following one of the four data flow charts shown in Fig. 5.

5 Experimental results

The simulation result of the proposed scaling engine was compared with the results from NN interpolation, linear interpolation, Kim's Winscale method (Kim *et al.*, 2003), B-spline approximation, and B-spline interpolation (Hou and Andrews, 1978). The test images in the LIVE database were used. The PSNR comparisons and the normalized cross-correlation coefficients (CCC) (Lehmann *et al.*, 1997) of luminance are listed in Table 3.

The CCC was used to assess image similarity and was calculated as follows:

$$\mathrm{CCC}=\left|\frac{\sum_{k,l}s(k,l)r(k,l)-KL\bar{s}\bar{r}}{\sqrt{\left(\sum_{k,l}s^2(k,l)-KL\bar{s}^2\right)\left(\sum_{k,l}r^2(k,l)-KL\bar{r}^2\right)}}\right|, \quad (8)$$

where $s(k,l)$ and $r(k,l)$ denote the original and the twice interpolated values in the point (k,l), respectively, and $\bar{s}$ and $\bar{r}$ denote the means of the original and the twice interpolated images of dimension KL, respectively.

The test images were processed in four groups. The scaling factor was 4/3, and the four different ways of scaling were the same as those described in Section 3.1. Since the NN interpolation simply replicates the nearest neighboring pixel, the images in Table 3 using the NN interpolation with respect to H up and V up are the same as the original images. Thus, the PSNR of the NN interpolation makes no sense and the mean of the CCC is 1, and the standard deviation (SD) is 0. As shown in the table, in terms of PSNR, Kim's Winscale, which is a method using an area pixel model, performs better than NN and linear interpolations. As an approximate kernel, B-spline approximation is slightly better than NN interpolation. Our proposed engine is slightly inferior to B-spline interpolation. However, the latter is too difficult to realize on hardware, because the coefficients of the B-spline interpolation kernel are achieved by prefiltering the data samples. Moreover, because the coefficients are theoretically infinite, it is impossible to implement. Our proposed engine performed much better than the other four methods. The mean value of the CCC was also verified by the PSNR evaluation.

Table 3 Peak signal-to-noise ratio (PSNR) of different test images scaled by factor 4/3

Method	PSNR (dB)*			
	H up & V up	H up & V down	H down & V up	H down & V down
NN		32.15	31.35	28.78
Linear	36.98	34.61	34.29	32.91
Winscale	37.72	35.23	34.89	33.47
B-spline approximation	33.44	31.91	31.70	30.67
B-spline interpolation	48.52	40.07	39.48	37.30
Proposed	46.24	39.71	39.09	36.92

Method	Cross-correlation coefficient							
	H up & V up		H up & V down		H down & V up		H down & V down	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
NN	1	0	0.9791	0.0416	0.9781	0.0415	0.9608	0.0725
Linear	0.9933	0.0129	0.9855	0.0331	0.9852	0.0331	0.9784	0.0501
Winscale	0.9927	0.0152	0.9848	0.0365	0.9844	0.0366	0.9776	0.0544
B-spline approximation	0.9797	0.0443	0.9726	0.0599	0.9722	0.0598	0.9659	0.0729
B-spline interpolation	0.9996	0.0009	0.9914	0.0272	0.9910	0.0273	0.9841	0.0489
Proposed	0.9986	0.0033	0.9906	0.0277	0.9902	0.0277	0.9834	0.0478

* Mean value. SD: standard deviation

From the statistical data of the CCC, especially from the SD of the six methods, we conclude that our proposed engine and the B-spline interpolation can give more stable scaling performance in all four scaling situations, because the SD of the cubic convolution and the B-spline interpolation are much smaller than those of the other four methods. Therefore, in our hardware implementation, we chose cubic convolution as the interpolation kernel.

To evaluate the proposed application specific integrated circuit (ASIC) implementation of the scaling engine, the MSEs from software based implementation and field-programmable gate array (FPGA) based implementation of test sequences were compared (Fig. 9). The test sequences used were Foreman, Tempete, and Mobile. In experiments, MSE comparisons of the luminance component Y and the chrominance components U and V were calculated. MSEs were all below 0.1 in these three test sequences. The results show that there was almost no difference between software simulation and hardware implementation of the proposed scaling engine.

It is difficult to measure the scaling performance exactly using only the PSNR, CCC, and MSE. Thus, in addition, we used subjective image quality evaluation to give an accurate evaluation. Fig. 10 shows the comparison of a subjective view processed by software based implementation and FPGA based implementation. Fig. 10a is the 16th frame of the Foreman test sequence which is 352×288 in YUV 4:2:0 CIF format. In the experiments, Fig. 10a is scaled up to 1024×768 through the proposed scaling engine. Fig. 10b is the snapshot of Fig. 10a. The image in Fig. 10c was processed by software based implementation with floating calculation. Fig. 10d was produced by the FPGA based scaling engine. Little difference can be seen between Figs. 10c and 10d. Thus, the proposed architecture and its hardware implementation of the scaling engine work very well according to the cubic-convolution scaling algorithm.

These experiments show that our proposed algorithm and its hardware implementation have theoretically better performance than the conventional algorithms, and that the image quality of our hardware implementation is nearly the same as that of our proposed algorithm.

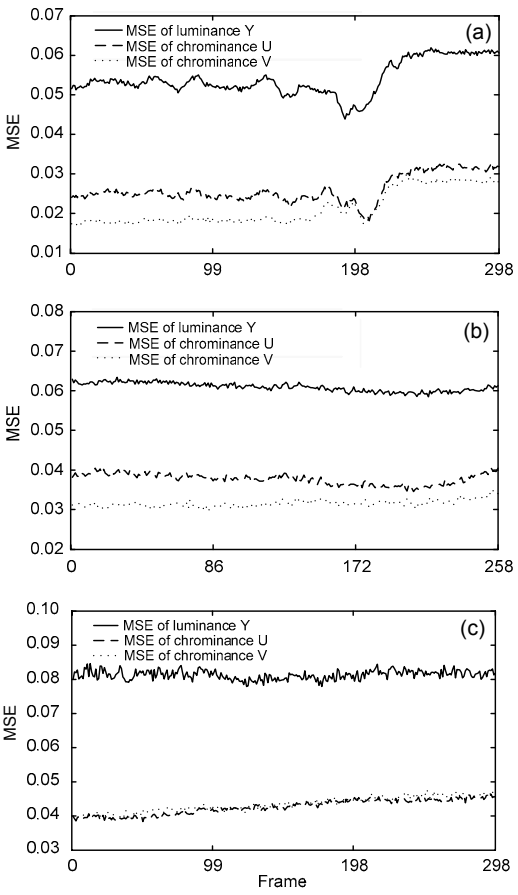

Fig. 9 Comparison of image quality between software based implementation and FPGA based implementation
(a) Foreman; (b) Tempete; (c) Mobile

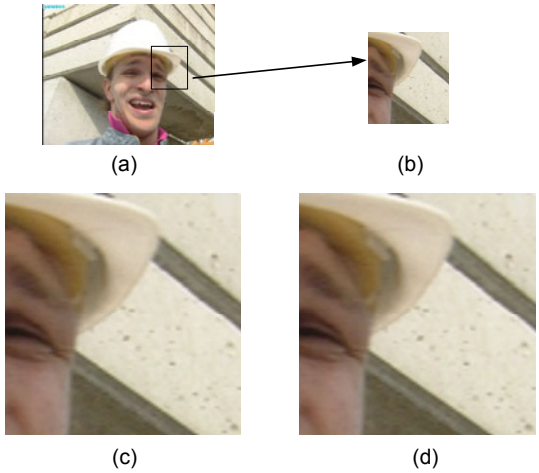

Fig. 10 Comparison of image quality between software based implementation and FPGA based implementation
(a) Original frame; (b) Snapshot; (c) Software based simulation; (d) FPGA based simulation

In addition, our proposed scaling engine was verified first on FPGA with the clock frequency at 130 MHz. Because of the limitation of the clock frequency of FPGA, the maximum resolution cannot reach our designed resolution of 2560×1920, but it can meet most of the demands of display devices. The scaling engine was also realized on 0.13 μm ASIC implementation. The clock frequency of ASIC design can reach 312 MHz, and our designed maximum resolution can be achieved.

The hardware cost of the proposed architecture has been compared with other FPGA based architectures (Feng *et al.*, 2001; Kim *et al.*, 2003; Aho *et al.*, 2007; Lin *et al.*, 2008; 2010) in Table 4. Our proposed implementation is one of the most complex implementations (Table 4). However, with the novel architecture, the gate count and the memory cost of our proposed implementation do not increase in proportion to the algorithm's computational complexity. The gate count of our proposed implementation is slightly larger than that of Aho *et al.* (2007), which is based on linear interpolation and is designed only for scaling down. The count is much smaller than that of any other implementations listed in the table. Aho *et al.* (2007) and Kim *et al.* (2003), whose algorithms are based on linear interpolation, used four line buffers to store the pixel data. Theoretically speaking, linear interpolation uses 2×2 pixels for interpolating, so the calculation needs two line buffers to export the interpolating pixel data, which means that their implementations need extra two line buffers to store and pre-process data. Our proposed implementation is based on a 4×4 cubic convolution. The interpolation needs at least four line buffers to deal with the vertical interpolation. Thus, except for these four line buffers, only one extra line buffer is used to store and pre-process data (Figs. 5 and 8). The theoretically smallest cost of line buffers that the 4×4 cubic convolution needs, is achieved by our proposed implementation. Other algorithms such as cubic convolution in the horizontal direction (Feng *et al.*, 2001) employ more line buffers than the proposed implementation.

A VLSI design of bi-cubic convolution was presented by Lin *et al.* (2008). Lin *et al.* (2010) also developed a piecewise linear convolution interpolation algorithm. The VLSI designs of Lin *et al*. (2008; 2010) are very efficient. The clock frequency of the 0.13 μm ASIC designs of Lin *et al.* (2008; 2010) can also reach 275 MHz as needed by our designed maximum resolution. However, the gate counts of their design are nearly double those of our design.

6 Conclusions

A novel scaling engine implementation is proposed in this paper, in which a 4×4 cubic convolution is employed to achieve better image quality. The engine is composed of only four functional units and five line buffers, which makes it more efficient in hardware cost than conventional methods. In experiments, some detailed comparisons and analyses indicated that the proposed implementation has a better performance than most of the existing scaling implementations. Furthermore, experiments showed that the strict fixed-point strategy can avoid cumulative errors and minimize the quantization errors in hardware implementation. As a result, our proposed engine is an effective image-scaling implementation for applications that require good image quality and low computational complexity.

Table 4 Comparison of FPGA based scaling implementation

Scaling algorithm	Scaling direction	Color format	Gate count	Line buffer	Clock frequency (MHz)
Bilinear (Aho *et al.*, 2007)	Down	YUV (4:2:0)	About 10 500	$4W_i$	105
Wincale (Kim *et al.*, 2003)	Up/Down	–	About 29 000	$4W_i$	
Cubic and bisigmoidal (Feng *et al.*, 2001)	Up	RGB	About 33 000	$6W_i$	46
Bi-cubic convolution (Lin *et al.*, 2008)	Up/Down	RGB/YUV	About 34 115	–	104
First-order polynomials convolution (Lin *et al.*, 2010)	Up/Down	RGB/YUV	About 28 904	–	104
Proposed	Up/Down	RGB/YUV	About 16 000	$5W_i$	130

W_i: input image width

References

Aho, E., Vanne, J., Hamalainen, T.D., Kuusilinna, K., 2007. Configurable implementation of parallel memory based real-time video downscaler. *Microprocess. Microsyst.*, **31**(5):283-292. [doi:10.1016/j.micpro.2006.09.003]

Arandiga, F., Donat, R., Mulet, P., 2003. Adaptive interpolation of images. *Signal Process.*, **83**(2):459-464. [doi:10.1016/S0165-1684(02)00445-0]

Chen, P.Y., Lien, C.Y., Lu, C.P., 2009. VLSI implementation of an edge-oriented image scaling processor. *IEEE Trans. VLSI Syst.*, **17**(9):1275-1284. [doi:10.1109/TVLSI.2008.2003003]

Erup, L., Gardner, F.M., Harris, R.A., 1993. Interpolation in digital modems. II. Implementation and performance. *IEEE Trans. Commun.*, **41**(6):998-1008. [doi:10.1109/26.231921]

Farrow, C.W., 1988. A Continuously Variable Digital Delay Element. IEEE Int. Symp. on Circuits and Systems, **3**:2641-2645. [doi:10.1109/ISCAS.1988.15483]

Feng, T., Xie, W.L., Yang, L.X., 2001. An Architecture and Implementation of Image Scaling Conversion. 4th Int. Conf. on ASIC, p.409-410. [doi:10.1109/ICASIC.2001.982587]

Gardner, F.M., 1993. Interpolation in digital modems. I. Fundamentals. *IEEE Trans. Commun.*, **41**(3):501-507. [doi:10.1109/26.221081]

Her, I., Yuan, C.T., 1994. Resampling on a pseudohexagonal grid. *CVGIP: Graph. Models Image Process.*, **56**(4):336-347. [doi:10.1006/cgip.1994.1030]

Hong, K.P., Paik, J.K., Kim, H.J., Lee, C.H., 1996. An edge-preserving image interpolation system for a digital camcorder. *IEEE Trans. Consum. Electron.*, **42**(3):279-284. [doi:10.1109/30.536121]

Hou, H., Andrews, H., 1978. Cubic splines for image interpolation and digital filtering. *IEEE Trans. Acoust. Speech Signal Process.*, **26**(6):508-517. [doi:10.1109/TASSP.1978.1163154]

Keys, R., 1981. Cubic convolution interpolation for digital image processing. *IEEE Trans. Acoust. Speech Signal Process.*, **29**(6):1153-1160. [doi:10.1109/TASSP.1981.1163711]

Kim, C.H., Seong, S.M., Lee, J.A., Kim, L.S., 2003. Winscale: an image-scaling algorithm using an area pixel model. *IEEE Trans. Circ. Syst. Video Technol.*, **13**(6):549-553. [doi:10.1109/TCSVT.2003.813431]

Lehmann, T., Sovakar, A., Schmitt, W., Repges, R., 1997. A comparison of similarity measures for digital subtraction radiography. *Comput. Biol. Med.*, **27**(2):151-167. [doi:10.1016/S0010-4825(97)83769-9]

Lehmann, T.M., Gonner, C., Spitzer, K., 1999. Survey: interpolation methods in medical image processing. *IEEE Trans. Med. Imag.*, **18**(11):1049-1075. [doi:10.1109/42.816070]

Li, X., Orchard, M.T., 2001. New edge-directed interpolation. *IEEE Trans. Image Process.*, **10**(10):1521-1527. [doi:10.1109/83.951537]

Lin, C.C., Sheu, M.H., Chiang, H.K., Liaw, C., Wu, Z.C., 2008. The Efficient VLSI Design of BI-CUBIC Convolution Interpolation for Digital Image Processing. IEEE Int. Symp. on Circuits and Systems, p.480-483. [doi:10.1109/ISCAS.2008.4541459]

Lin, C.C., Sheu, M.H., Liaw, C., Chiang, H.K., 2010. Fast first-order polynomials convolution interpolation for real-time digital image reconstruction. *IEEE Trans. Circ. Syst. Video Technol.*, **20**(9):1260-1264. [doi:10.1109/TCSVT.2010.2057017]

Nuno-Maganda, M.A., Arias-Estrada, M.O., 2006. Real-Time FPGA-Based Architecture for Bicubic Interpolation: an Application for Digital Image Scaling. Int. Conf. on Reconfigurable Computing and FPGAs, p.1-8. [doi:10.1109/RECONFIG.2005.34]

Parker, J.A., Kenyon, R.V., Troxel, D.E., 1983. Comparison of interpolating methods for image resampling. *IEEE Trans. Med. Imag.*, **2**(1):31-39. [doi:10.1109/TMI.1983.4307610]

Sheikh, H.R., Sabir, M.F., Bovik, A.C., 2006. A statistical evaluation of recent full reference image quality assessment algorithms. *IEEE Trans. Image Process.*, **15**(11):3440-3451. [doi:10.1109/TIP.2006.881959]

Sheikh, H.R., Wang, Z., Cormack, L., Bovik, A.C., 2010. LIVE Image Quality Assessment Database Release 2. Available from http://live.ece.utexas.edu/research/quality/subjective.htm [Accessed on Oct. 18, 2010].

Shi, H.J., Ward, R., 2002. Canny Edge Based Image Expansion. IEEE Int. Symp. on Circuits and Systems, **1**:785-788. [doi:10.1109/ISCAS.2002.1009958]

Shi, J.Z., Reichenbach, S.E., 2006. Image interpolation by two-dimensional parametric cubic convolution. *IEEE Trans. Image Process.*, **15**(7):1857-1870. [doi:10.1109/TIP.2006.873429]

Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P., 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, **13**(4):600-612. [doi:10.1109/TIP.2003.819861]