Alternatives to Bicubic Interpolation Considering FPGA Hardware Resource Consumption

Seyfeddine Boukhtache, Benoit Blaysat, Michel Grédiac, and Francois Berry

***Abstract*— Bicubic interpolation is widely used in real-time image processing systems because of its quality. The real-time implementation of bicubic interpolation requires a lot of hardware resources, especially the number of multipliers because it represents high computational complexity. In this article, a set of algorithms that approximate the bicubic interpolation and reduce the hardware resource consumption are proposed. The proposed algorithms are based on combining linear and cubic interpolations. These algorithms are surveyed and compared in terms of interpolation quality, number of adders, number of multipliers, adaptive logic modules, lookup tables (LUTs), registers, and maximum operating frequency. These algorithms are implemented and tested on an Intel Cyclone V target. This article provides various choices of interpolation algorithms to cater to different application requirements, including accuracy, hardware resource consumption, and throughput performance. The implementation codes are available at `github.com/DreamIP/Interpolation`.**

***Index Terms*— Bicubic interpolation, field-programmable gate array (FPGA), hardware precision, hardware resource consumption, interpolation quality.**

I. Introduction

NOWADAYS, a continuous increase use of digital image processing can be observed in both commercial and industrial applications. Interpolation plays a crucial role in digital image processing. It defines the accuracy and may affect the processing time of the target application. It is widely used and required by many real-time applications, such as resolution enhancement, image resizing, and geometric transformation, or in various domains, such as displacement estimation with digital image correlation. Interpolation retrieves infinite resolution data from a set of discrete inputs. Image interpolation aims at computing image value at any (subpixel) location, by relying on the neighbor pixels intensities and location.

The accuracy of the obtained results depends on the interpolation quality. The highest performance of interpolation is primordial in order to reach the desired requirements of the target application, especially accuracy. The higher the interpolation quality, the more the hardware resources required for its implementation and, thus, the higher the design cost, as this cost heavily relies on the use of field-programmable gate array (FPGA) hardware resources.

An optimal architecture represents the best tradeoff between accuracy and hardware cost. Consequently, interpolation must be carefully studied in terms of output quality, hardware resource consumption, and throughput performance for practical applications.

The quality of the results provided by interpolation may significantly change according to the used algorithm. Many interpolation algorithms are available in the literature, and choosing the one offering the best tradeoff between computational complexity, processing time, and interpolation quality is crucial to fulfill real-time requirements and to reach hardware efficiency for applications based on interpolation.

Among the existing interpolation algorithms, the following ones are the most commonly used. With the nearest neighbor (NN) method, the interpolation model is a zero-order polynomial. It is the simplest and fastest algorithm. However, its main drawback is to induce the highest error. Bilinear interpolation is rapid. It relies on a first-order polynomial model, which is simple to implement. It is more accurate than NN, but interpolation quality can still be improved [1]–[3]. Winscale interpolation [4], [5] is a linear scaling algorithm. It delivers better image quality than the bilinear algorithm. Its hardware cost is low, but the resulting images involve undesirable effects. Cubic spline interpolation involves high computational complexity (i.e., resolving matrix problems) in order to achieve higher quality. Thus, its hardware implementation is not simple and requires a lot of processing time. Bicubic interpolation is performed by using a third-order polynomial function. It features better quality than the Winscale interpolation, and it is less complex to implement than cubic spline interpolation [1], [2], [6]. The higher the order of the kernel function, the better the interpolation quality. Also, the throughput performance is inversely proportional to the kernel order. Lanczos interpolation consists in using a lookup table (LUT) of a windowed-sinc function [7]. Of course, its accuracy depends on the discretization step; in other words, on memory utilization. Lanczos 2 and 3 require 16 and 36 neighbor pixels, respectively. Lanczos 2 offers almost a similar quality than bicubic interpolation. Lanczos 3 offers higher interpolation quality. It requires ten more multipliers and adders to perform a 2-D interpolation compared with Lanczos 2. The kernels of the interpolation methods discussed earlier are listed in Table I.

Manuscript received July 8, 2020; revised September 23, 2020; accepted October 3, 2020. Date of publication November 16, 2020; date of current version January 28, 2021. This work was supported in part by the French government research program "Investissements d'Avenir" through the IDEX-ISITE initiative 16-IDEX-0001 under Grant CAP 20-25; and in part by the IMobS3 Laboratory of Excellence under Grant ANR-10-LABX-16-01. *(Corresponding author: Francois Berry.)*

The authors are with the Université Clermont Auvergne, CNRS, SIGMA Clermont, Institut Pascal, F-63000 Clermont-Ferrand, France (e-mail: seyfeddine.boukhtache@uca.fr; benoit.blaysat@uca.fr; michel.grediac@uca.fr; francois.berry@uca.fr).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2020.3032888

TABLE I
KERNEL OF THE INTERPOLATION TECHNIQUES

Method	NN	Bilinear	Bi-cubic	spline	Lanczos
Kernel type	0-order	1st-order function	3rd-order function	nth-order function	Look-Up Table

Furthermore, various interpolation algorithms based on convolutional neural network (CNN) are used for super-resolution purpose. In the works of Dong *et al.* [8], [9], the input image is enlarged in advance to the desired output size by using the bicubic interpolation. It is then given to the CNN. Kim *et al.* [10] proposed deeply recursive CNNs that outperform the work of Dong *et al.* [8], [9] in terms of quality. However, deeply recursive architecture is not suitable for real-time implementation on FPGAs. Shi *et al.* [11] reconstructed the output image by using multiple input images, but this increases the computational complexity. Manabe *et al.* [12] implemented a CNN-based architecture on an FPGA to provide super-resolution images in real time. CNN-based algorithms offer high interpolation quality at the price of very high computational complexity, which makes them not suitable for FPGA implementation. This is the reason why bicubic interpolation is considered as the technique leading to the best tradeoff between interpolation quality and computational complexity or hardware resource consumption compared with the other algorithms [1], [6]. Bicubic interpolation is also successfully used for various applications, such as image scaling [13], image quality enhancement [14], and super-resolution [15].

In this work, four algorithms are proposed in order to reduce the consumption of hardware resources while keeping the desired bicubic interpolation quality. These algorithms reduce the use of multipliers by more than 65%, require 30% less of ALMs and registers compared with the bicubic algorithm, and approximate at best the bicubic interpolation.

This article is organized as follows. Section II briefly recalls the basics of bicubic interpolation. Section III surveys previous hardware implementations of the bicubic algorithm, which are available in the literature. The approximation of the third-order cubic kernel by n-piecewise linear functions is detailed in Section IV. Section V introduces the combinations based on linear and cubic interpolations. Section VI presents and compares the interpolation quality and the hardware implementation results of these algorithms.

II. BICUBIC INTERPOLATION

Bicubic interpolation is the natural extension of 1-D cubic interpolation to construct new data on a 2-D space. The 1-D interpolation function can be written as follows:

$$g(x) = \sum_{k=0}^{3} A_k \beta(x - x_k) = \sum_{k=0}^{3} A_k \beta(u_k) \tag{1}$$

where A_k, $k = 0 \ldots 3$, are the set of discrete data, β is the interpolation kernel, and g is the output, i.e., the interpolated function.

Keys [6] used the cubic interpolation kernel given in (2). It requires four neighbor pixels

$$\beta(u) = \begin{cases} (a+2)|u|^3 - (a+3)|u|^2 + 1, & 0 \le |u| < 1 \\ a|u|^3 - 5a|u|^2 + 8a|u| - 4a, & 1 \le |u| < 2 \\ 0, & \text{others.} \end{cases} \tag{2}$$

Parameter a is set to -0.5 in order to reach and to offer the best interpolation quality [6]. This leads to the following expression of $\beta(u)$:

$$\beta(u) = \begin{cases} \frac{3}{2}|u|^3 - \frac{5}{2}|u|^2 + 1, & 0 \le |u| < 1 \\ -\frac{1}{2}|u|^3 + \frac{5}{2}|u|^2 - 4|u| + 2, & 1 \le |u| < 2 \\ 0 & \text{others.} \end{cases} \tag{3}$$

Bicubic interpolation relies on 16 neighboring pixels. Two types of 1-D interpolations are performed in turn along the horizontal and the vertical directions, as shown in Fig. 1.

It can be seen that the vertical interpolation is performed four times to obtain four intermediate points (q_0, q_1, q_2, q_3), whereas the horizontal one, which is based on these intermediate points, is performed once. It is worth noting that the order between the two directions (vertical and horizontal) can be switched without any impact on the value obtained at the end of the procedure.

The bicubic formula is given by

$$g(x, y) = \sum_{i=0}^{3} \Big(\sum_{j=0}^{3} A_{ij} \beta(y - y_j) \Big) \beta(x - x_i). \tag{4}$$

The preceding equation can be rewritten with $u_i = x - x_i$ and $v_i = y - y_i$. Thus

$$\begin{aligned} & g(x, y) \\ &= (\beta(v_0)A_{00} + \beta(v_1)A_{10} + \beta(v_2)A_{20} + \beta(v_3)A_{30})\beta(u_0) \\ &\quad + (\beta(v_0)A_{01} + \beta(v_1)A_{11} + \beta(v_2)A_{21} + \beta(v_3)A_{31})\beta(u_1) \\ &\quad + (\beta(v_0)A_{02} + \beta(v_1)A_{12} + \beta(v_2)A_{22} + \beta(v_3)A_{32})\beta(u_2) \\ &\quad + \beta(v_0)A_{03} + \beta(v_1)A_{13} + \beta(v_2)A_{23} + \beta(v_3)A_{33})\beta(u_3) \end{aligned} \tag{5}$$

where $\beta(u_k)$ and $\beta(v_k)$ are the coefficients used for the horizontal and vertical interpolations, respectively. They are calculated by using (3).

III. PREVIOUS WORKS

This section is a brief overview of previous hardware implementations of the bicubic algorithm presented earlier. These previous hardware implementations can be classified into two categories.

1) Heterogeneous architecture is the standard one. It can interpolate at different locations, as shown in Fig. 2(a). Indeed, different coefficients are recalculated for each location. This architecture can be used by any application, such as image rotations and deformation measurements.
2) Homogeneous architecture is a special case of the heterogeneous one [see Fig. 2(b)]. It is suited only when interpolation is performed at the same location for all windows of the image. This means that the coefficients

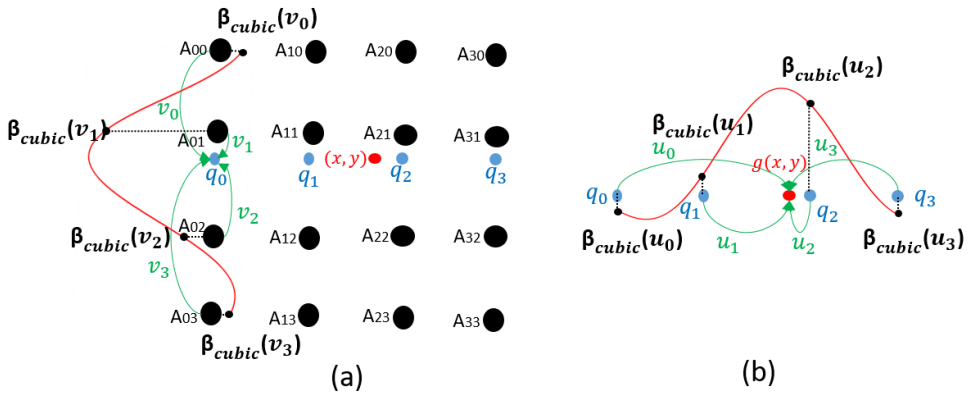

Fig. 1. Bicubic interpolation. (a) Vertical and (b) horizontal 1-D interpolations.

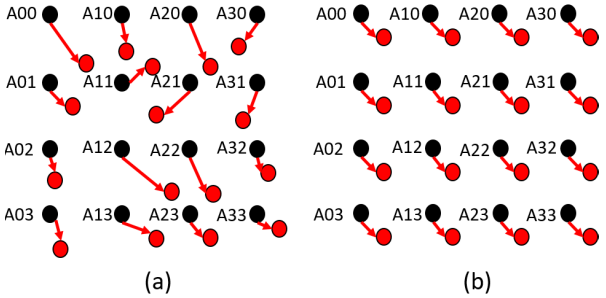

Fig. 2. (a) Heterogeneous and (b) homogeneous interpolations. Red points: locations of interpolation.

TABLE II
CLASSIFICATION OF BICUBIC IMPLEMENTATIONS AVAILABLE IN THE LITERATURE

Work	Architectural features
Homogeneous	
Lin *et al.* [16]	10 multipliers and 437 Logic Blocks. ASIC : 30k gates.
Lin *et al.* [17]	8 multipliers and 376 Logic Blocks. ASIC : 26k gates.
Lin *et al.* [18]	8 multipliers and 414 Logic Blocks. ASIC : 28k gates.
Wang *et al.* [19]	6 multipliers. ASIC : 16k gates.
Gour, *et al.* [20]	8 multipliers.
Heterogeneous	
Nuno *et al.* [21]	32 multipliers and 890 Logic Blocks.
Liu *et al.* [22]	36 multipliers.
Zhang *et al.* [23]	20 multipliers.

are calculated only one time because the same ones will be used for all the pixels, such as in image scaling. The existing CNN-based interpolations are based on homogeneous architecture.

Bicubic implementations available in the literature are reported in Table II.

The heterogeneous architecture is more complex and calls for more hardware resources. Indeed, it consists in implementing five 1-D-cubic interpolations instead of the two 1-D-cubic interpolations required in the homogeneous case. This article considers the heterogeneous case.

Nuno-Maganda and Arias-Estrada [21] decomposed the bicubic algorithm into two main blocks. The first one generates the interpolation coefficients, and the second one performs the bicubic interpolation (5). The bicubic formula given in (5) is implemented in four parallel subblocks, each subblock representing one of the four lines of this equation. The design was implemented on a Virtex-II Pro FPGA and operated at a maximum operating frequency of 100 MHz; 32 multipliers and 890 logic blocks are necessary in this case.

Liu *et al.* [22] also implemented a heterogeneous bicubic interpolation. They simplified the calculation of the interpolation coefficients according to the relationship between them. Their architecture required 36 multipliers to perform interpolation, making it more complex than [21].

In order to reduce the computational complexity, Zhang *et al.* [23] divided the interval between two neighboring pixels into eight subintervals. Then, the coefficients for each of these subintervals were computed and stored to be used each time interpolation was performed. The accuracy of this method depends on the number of subintervals. A drawback is that this architecture is counterbalanced by a poor interpolation quality and high memory utilization. The hardware resource consumption of the proposed design is not reported by the authors.

A homogeneous architecture of the bicubic interpolation is implemented in [16], [17], [19], and [20]. The architecture of [16] and [19] is based on the exact cubic kernel. While an approximated kernel is used in [17], [18], and [20] to reduce the hardware resource consumption, the cubic kernel is approximated in [17] and [20] by using two and four-piecewise linear functions, respectively. The authors applied the piecewise linear functions in order to approximate a simpler cubic kernel. They use a kernel with "$a = -1$" instead of "$a = -0.5$," which leads to reduce not only the complexity of the cubic kernel but also the interpolation quality. Gour *et al.* [20] applied four-piecewise linear functions, but they did not perform a hardware implementation. Lin *et al.* [18] presented an approximation of the ideal

sinc-function over the $[-2, 2]$ interval with four-piecewise linear functions.

In this work, we propose a set of algorithms that reduce the hardware resource consumption of the heterogeneous architecture and accurately approximate bicubic interpolation. First, we extend the idea of approximating the interpolation kernel with n-piecewise linear functions used in [17], [18], and [20] to the heterogeneous architecture. The third-order cubic kernel ($a = -0.5$) is approximated with two-, four-, and six-piecewise linear functions. Then, we combine the 1-D cubic and 1-D linear interpolations to obtain the 2-D interpolation with less hardware resource consumption.

IV. Approximation of the Cubic Kernel With n-Piecewise Linear Functions

A piecewise function is a function in which more than one formula is used to define the output over different pieces or segments of the domain. Each formula is valid over its domain, and the domain of the function is the union of all these small domains. Lin *et al.* [17], [18] and Gour *et al.* [20] [17], [18], [20] approximated the interpolation kernel using two- and four-piecewise linear functions and addressed a homogeneous architecture, as described in the previous section.

In this section, the third-order cubic kernel ($a = -0.5$) is approximated with two-, four-, and six-piecewise linear functions. Furthermore, the coefficients of each case are adapted such that the lowest hardware resource consumption is obtained. First, coefficients that can be accurately represented in fixed point are provided. These new coefficients maintain almost the same quality of the approximation. Then, the use of multipliers when multiplied by a constant value is avoided by using constants equal to a power of 2. The adapted hardware kernels are given in (6)–(8) and Fig. 3

$$\beta_2(x) = \begin{cases} -|x|+1, & 0 \le |x| < 1 \\ -0.03125|x|+0.03125, & 1 \le |x| < 2 \\ 0 & \text{others} \end{cases} \tag{6}$$

$$\beta_4(x) = \begin{cases} -0.25|x|-0.125|x|+1, & 0 \le |x| < 0.17 \\ -0.125|x|-|x|+1.125, & 0.17 \le |x| < 1 \\ -0.25|x|+0.25, & 1 \le |x| < 1.33 \\ 0.125|x|-0.25, & 1.33 \le |x| < 2 \\ 0 & \text{others} \end{cases} \tag{7}$$

$$\beta_6(x) = \begin{cases} -0.5|x|+1, & 0 \le |x| < 0.25 \\ -0.25|x|-|x|+1.1875, & 0.25 \le |x| < 0.875 \\ -0.5|x|-0.25|x|+0.75, & 0.875 \le |x| < 1 \\ -0.25|x|-0.0625|x|+0.3125, & 1 \le |x| < 1.218 \\ -0.0625|x|+0.0156, & 1.218 \le |x| < 1.375 \\ 0.125|x|-0.25, & 1.375 \le |x| < 2 \\ 0 & \text{others} \end{cases} \tag{8}$$

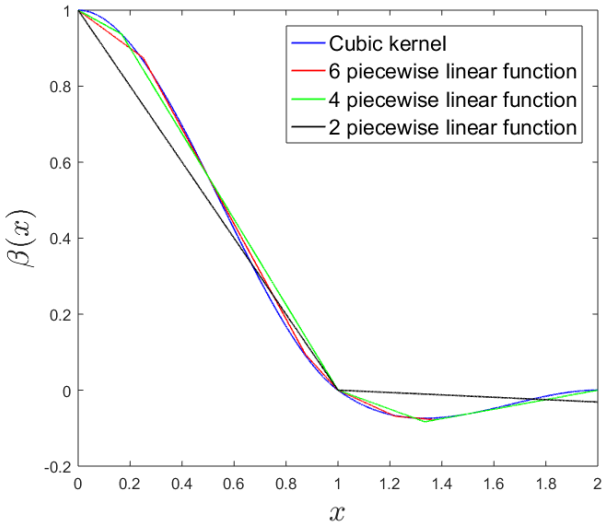

Fig. 3. Approximation of the cubic kernel based on n-piecewise linear functions.

TABLE III
Comparison Between Piecewise Linear Algorithms

Algorithm	Multipliers	Adders	MAE
2-piecewise	5	25	0.0463
4-piecewise	20	39	0.0163
6-piecewise	20	39	0.0081

The mean absolute error (MAE) is determined in order to evaluate and compare the approximations based on different numbers of piecewise linear functions. It is based on the following equation:

$$\text{MAE}_j = \frac{1}{N}\sum_{i=1}^{N} |\beta(x_i) - \beta_j(x_i)|. \tag{9}$$

Implementing a heterogeneous architecture is based on instantiating the 1-D architecture five times, as described in Section II. Four 1-D interpolations are used in order to perform the 1-D interpolation in the first direction. The outputs of these four previous 1-D interpolations provide the inputs of the interpolation in the second direction. The characterizations of each algorithm are summarized in Table III.

The two-piecewise linear functions algorithm requires only five multipliers and 25 adders. It reduces the hardware consumption compared with the two-piecewise linear functions proposed in [17]. However, it features a higher MAE than the four- and six-piecewise linear functions. The six-piecewise linear functions algorithm provides a better quality of interpolation for the same hardware price as the four-piecewise linear functions; 20 multipliers and 39 adders are required in both cases.

Combinations of linear and cubic interpolations are proposed in Section V in order to reach better performances.

V. Combining Cubic and Linear Interpolations

We propose to combine the 1-D linear and cubic interpolations in order to obtain a 2-D interpolation, which features

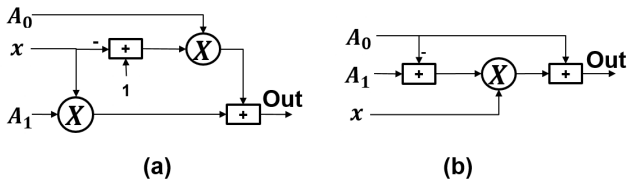

Fig. 4. Block diagram of (a) linear interpolation and (b) simplified linear interpolation.

TABLE IV
COEFFICIENTS OF THE SIMPLIFIED CUBIC INTERPOLATION

Coefficient	Expression
t_0	A_1
t_1	$0.5A_2 - 0.5A_0$
t_2	$A_0 - 2.5A_1 + 2A_2 - 0.5A_3$
t_3	$-0.5A_0 + 1.5A_1 - 1.5A_2 + 0.5A_3$

lower hardware cost and keeps an interpolation quality of the same order as the bicubic one. Consequently, this section first presents the simplified architecture of linear and cubic interpolations. Then, all their possible combinations are detailed and described in turn.

Linear interpolation is simple and easy to implement. It is based on a first-order interpolation kernel. It requires only two pixels to perform the 1-D interpolation. Furthermore, the linear kernel as shown in (10) does not require any intensive calculations. However, it is lower in terms of interpolation quality compared with the cubic one

$$\beta_{\text{linear}}(x) = \begin{cases} 1 - |x|, & 0 \le |x| < 1 \\ 0, & \text{others} \end{cases}. \tag{10}$$

By substituting (10) into (1), we obtain the 1-D linear interpolation given by the following equation:

$$g_{\text{linear}}(x) = A_0 \times (1 - x) + A_1 \times x. \tag{11}$$

Equation (11) is simplified by a simple factorization in order to reduce the number of multipliers from 2 to 1, as shown in (12) and Fig. 4

$$g_{\text{linear}}(x) = (A_1 - A_0) \times x + A_0. \tag{12}$$

Cubic interpolation is implemented by using the same principle of factorization in order to reduce the number of multipliers. The obtained formula of cubic interpolation is given in (13), where coefficients (t_0, t_1, t_2, t_3) are obtained by using the zero-order function that depends only on the neighbor pixels (A_0, A_1, A_2, A_3) (see Table IV). Thus

$$g_{\text{cubic}}(x) = t_0 + \big(t_1 + (t_2 + t_3 \times x) \times x\big) \times x. \tag{13}$$

The architecture of the simplified cubic interpolation is shown in Fig. 5.

Combining linear and cubic interpolations leads to three possibilities: applying one cubic interpolation and four linear interpolations, two cubic interpolations and one linear interpolation, and three cubic interpolations and two linear interpolations. Another algorithm based on three cubic interpolations and a modified linear interpolation is also presented.

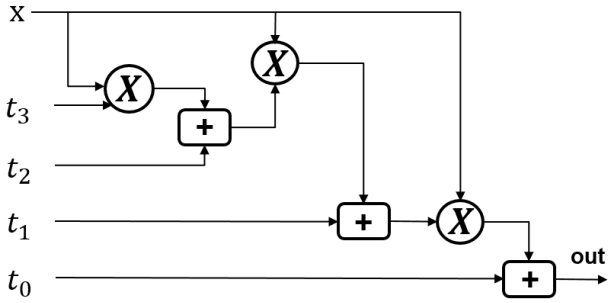

Fig. 5. Block diagram of the simplified cubic interpolation.

Combining two different interpolation kernels may lead to obtaining an anisotropic algorithm. In other words, the order of the 1-D interpolations (horizontal than vertical or vertical than horizontal) is an important factor. Consequently, it is of prime importance to study the isotropy or the anisotropy of these algorithms.

The isotropy of the proposed algorithms based on linear and cubic combinations is studied analytically. This is done by comparing the analytic interpolation expressions of a reference interpolant with its transpose. The error due to the anisotropy is assessed with a quantity named "E_{Isotropy}" defined in (14). An algorithm is called isotropic if "E_{Isotropy}" is null

$$E_{\text{Isotropy}} = |g(x, y, A_{ij}) - g(y, x, A_{ji})|. \tag{14}$$

The following namings are used in order to study the error due to the anisotropy of the proposed algorithms. β_{linear} and β_{cubic} are the linear and cubic kernels given by (3) and (10), respectively. (x_0, x_1, x_2, x_3) and (y_0, y_1, y_2, y_3) replace the expressions $(1+x, x, 1-x, 2-x)$ and $(1+y, y, 1-y, 2-y)$, respectively.

The hardware resource consumption according to the requested hardware precision is also studied based on interval arithmetic analysis, such as in [24]. In this analysis, the worst case is used for evaluating the error due to the fixed-point representation (bit width) at the output of each arithmetic operation. It is important to note that this error is cumulative with the one due to the approximation of the interpolation kernel. In this analysis, the following parameters are used.

1) "m" is the pixel data length $(A_{00}, A_{01}, \ldots, A_{33})$.
2) "k" defines the length of the inputs of the multipliers.
3) "$k - m - 1$" defines the bit width of the fractional part.

The proposed algorithms are described and detailed in the following.

A. One Cubic and Four Linear Interpolation

In this algorithm, we propose to apply a linear interpolation in the first direction and a cubic one in the second direction. Thus, the 2-D interpolation is performed by applying first four linear interpolations in the first direction in order to obtain the four intermediate points (S_0, S_1, S_2, and S_3).

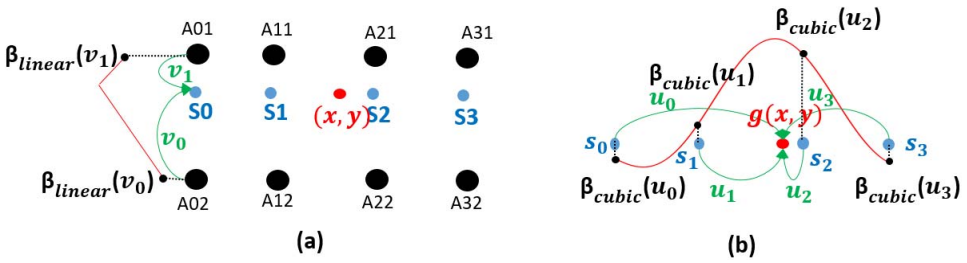

Fig. 6. Combination of linear and cubic interpolations. (a) Four linear interpolations. (b) One cubic interpolation.

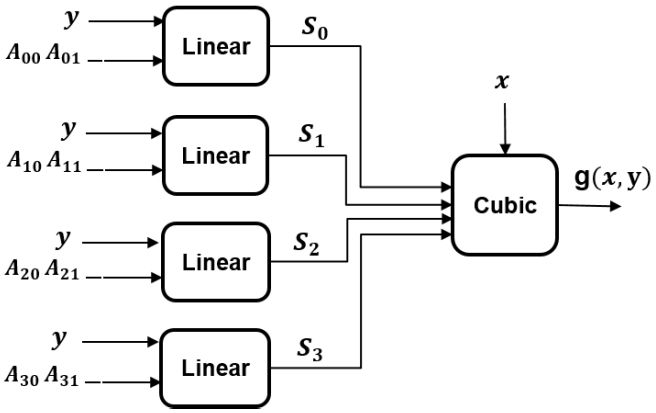

Fig. 7. Block diagram of four linear interpolations and one cubic interpolation.

These intermediate points are then used to perform a cubic interpolation in the second direction, as shown in Fig. 6.

Linear interpolation requires only two pixels to interpolate intermediate locations by linearly connecting them. Consequently, 2-D interpolation requires only eight pixels to perform the four linear interpolations instead of 16 pixels in the case of bicubic interpolation. This leads to less memory requirement as only one image line and four pixels values are required compared to standard bicubic interpolation with three lines and four pixels. Furthermore, the architecture is based on a simplified linear and cubic interpolation, as shown in Fig. 7. This leads to an optimal architecture in terms of hardware resource, especially the number of multipliers.

The architectures of "Linear" and "Cubic" blocks are shown in Figs. 4(b) and 5, respectively. The hardware architecture of four linear and one cubic interpolation requires eight neighbor pixels, seven multipliers, and 21 adders.

The isotropy of this algorithm is studied in the following. The 2-D interpolation can be rewritten as follows:

$$\begin{aligned} &g(x, y, A_{ij}) \\ &= [\beta_{\text{linear}}(y_1)A_{01} + \beta_{\text{linear}}(y_2)A_{02}]\beta_{\text{cubic}}(x_0) \\ &\quad +[\beta_{\text{linear}}(y_1)A_{11} + \beta_{\text{linear}}(y_2)A_{12}]\beta_{\text{cubic}}(x_1) \\ &\quad +[\beta_{\text{linear}}(y_1)A_{21} + \beta_{\text{linear}}(y_2)A_{22}]\beta_{\text{cubic}}(x_2) \\ &\quad +[\beta_{\text{linear}}(y_1)A_{31} + \beta_{\text{linear}}(y_2)A_{32}]\beta_{\text{cubic}}(x_3). \end{aligned} \tag{15}$$

Note that $\beta_{\text{linear}}(y_1) + \beta_{\text{linear}}(y_2) = 1$. The A_{ij} parameters are maximized when its coefficient is positive and minimized when it is negative in order to reach the worst case.

TABLE V
MAXIMUM HARDWARE ERROR VERSUS MULTIPLIER SIZE (k) AND PIXEL DATA LENGTH (m)

Block	Maximum error
Linear	$2 \times 2^{m+1-k}$
Cubic	$\mathbf{15 \times 2^{m+1-k}}$

These leads to the following equation:

$$\begin{aligned} &\text{Isotropy}_{\text{error}} \\ &= |g(x, y, A_{ij}) - g(y, x, A_{ji})| \\ &\le \max(A_{ij})[\beta_{\text{cubic}}(x_1) + \beta_{\text{cubic}}(x_2) - \beta_{\text{cubic}}(y_1) - \beta_{\text{cubic}}(y_2)] \\ &\quad + \min(A_{ij})[\beta_{\text{cubic}}(x_0) + \beta_{\text{cubic}}(x_3)] \\ &\quad + \max(A_{ij})[-\beta_{\text{cubic}}(y_0) - \beta_{\text{cubic}}(y_3)]. \end{aligned} \tag{16}$$

Developing (16) leads to obtain the maximum error, where "m" is the pixel data length in bits

$$\text{Isotropy}_{\text{error}} \le 0.125 \times 2^m. \tag{17}$$

The error due to the hardware representation (bit width) is studied based on the procedure described earlier. Table V gives the maximum hardware error versus hardware resource consumption for each block of this architecture.

The global interpolation error for this architecture consists of the error of the last block, which is $E_{\text{inter}} = E_{\text{cubic}} = 15 \times 2^{m+1-k}$.

B. Two Cubic and One Linear Interpolation

In this algorithm, the cubic interpolation is applied in the first direction and the linear interpolation in the other direction. Hence, two cubic interpolations are performed first by using eight pixels to obtain two intermediate points (S_0, S_1). These two intermediate points are required to perform the linear interpolation in the second direction, as shown in Fig. 8.

The architecture of this algorithm is presented in Fig. 9. It is based on the simplified linear and cubic interpolations described in Figs. 4 and 5.

The hardware implementation of the presented algorithm requires seven multipliers and 28 adders. Furthermore, as the previous algorithm, only eight neighbor pixels are required. This algorithm consumes seven more adders than the algorithm combining one cubic and four linear interpolations.

A similar method is also applied for this algorithm in order to evaluate its isotropy or anisotropy. The same maximum error

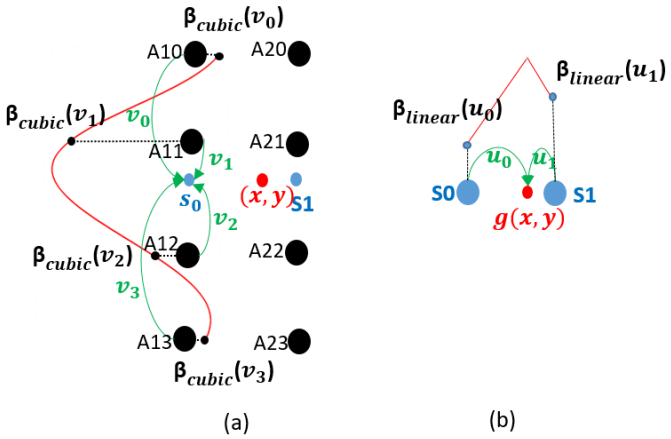

Fig. 8. Combination of linear and cubic interpolations. (a) Two cubic and (b) one linear interpolation.

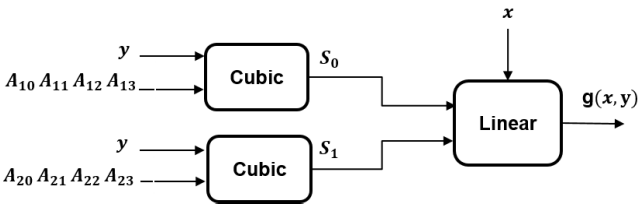

Fig. 9. Block diagram of two cubic and linear interpolations.

TABLE VI
MAXIMUM HARDWARE ERROR VERSUS MULTIPLIER SIZE (k) AND PIXEL DATA LENGTH (m)

Block	Maximum error
Cubic	$7 \times 2^{m+1-k}$
Linear	$\mathbf{16 \times 2^{m+1-k}}$

is obtained because, in this algorithm, only the order of the 1-D interpolations is reversed.

The maximum hardware error versus resource consumption for each block is given in Table VI.

The global interpolation error for this architecture consists of the error of the last block, which is $E_{\text{inter}} = E_{\text{Linear}} = 16 \times 2^{m+1-k}$.

C. Three Cubic and Two Linear Interpolation

In the present case, the four intermediate points (S_0–S_3) are obtained by performing two cubic and two linear interpolations, as shown in Fig. 10. Since the interesting points are the two points, which are the nearest to the interpolation location (S_1 and S_2), two cubic interpolations are applied to obtain them and to keep the desired quality of the interpolation. Points S_0 and S_3 are obtained by two linear interpolations to reduce the hardware resource utilization. Then, a 1-D cubic interpolation is applied in the other direction to perform the 2-D interpolation without impairing the interpolation quality.

The hardware architecture of this algorithm is shown in Fig. 11. It requires 12 neighbor pixels: eight pixels to perform the two cubic interpolations and four pixels to perform the two linear interpolations in the first direction. It consumes 11 multipliers and 43 adders to perform the 2-D interpolation.

TABLE VII
MAXIMUM HARDWARE ERROR VERSUS MULTIPLIER SIZE (k) AND PIXEL DATA LENGTH (m)

Block	Maximum error
Linear#1	$2 \times 2^{m+1-k}$
Cubic#1	$7 \times 2^{m+1-k}$
Cubic#2	$\mathbf{25 \times 2^{m+1-k}}$

The isotropy of this algorithm is addressed in the following. The 2-D interpolation based on three cubic and two linear 1-D interpolations is described by the following equation:

$$\begin{aligned} & g(x, y, A_{ij}) \\ &= [\beta_{\text{linear}}(y_1)A_{01} + \beta_{\text{linear}}(y_2)A_{02}]\beta_{\text{cubic}}(x_0) \\ &\quad + [\beta_{\text{cubic}}(y_0)A_{10} + \beta_{\text{cubic}}(y_1)A_{11} + \beta_{\text{cubic}}(y_2)A_{12} \\ &\quad + \beta_{\text{cubic}}(y_3)A_{13}]\beta_{\text{cubic}}(x_1) \\ &\quad + [\beta_{\text{cubic}}(y_0)A_{20} + \beta_{\text{cubic}}(y_1)A_{21} + \beta_{\text{cubic}}(y_2)A_{22} \\ &\quad + \beta_{\text{cubic}}(y_3)A_{23}]\beta_{\text{cubic}}(x_2) \\ &\quad + [\beta_{\text{linear}}(y_1)A_{31} + \beta_{\text{linear}}(y_2)A_{32}]\beta_{\text{cubic}}(x_3). \end{aligned} \tag{18}$$

The error due to anisotropy is given by

$$\begin{aligned} & E_{\text{Isotropy}} \\ &= |g(x, y, A_{ij}) - g(y, x, A_{ji})| \\ &\leq \min(A_{ij})\beta_{\text{cubic}}(y_0)[\beta_{\text{cubic}}(x_1) + \beta_{\text{cubic}}(x_2) - 1] \\ &\quad + \min(A_{ij})\beta_{\text{cubic}}(y_3)[\beta_{\text{cubic}}(x_1) + \beta_{\text{cubic}}(x_2) - 1] \\ &\quad + \max(A_{ij})\beta_{\text{cubic}}(x_0)[1 - \beta_{\text{cubic}}(y_1) - \beta_{\text{cubic}}(y_2)] \\ &\quad + \max(A_{ij})\beta_{\text{cubic}}(x_3)[1 - \beta_{\text{cubic}}(y_1) - \beta_{\text{cubic}}(y_2)]. \end{aligned} \tag{19}$$

Developing the previous equation leads to

$$E_{\text{Isotropy}} \leq 0.0156 \times 2^m. \tag{20}$$

The maximum hardware error versus the resource consumption for each block is given in Table VII.

The global interpolation error for this architecture consists of the error of the last block, which is $E_{inter} = E_{\text{Cubic\#2}} = 25 \times 2^{m+1-k}$.

D. Three Cubic and Two Modified Linear Interpolation

Using three cubic interpolations better maintains the interpolation quality compared with one and two cubic interpolations. Consequently, we propose in this algorithm to fix the number of 1-D cubic interpolations to three. This is obtained by applying two 1-D cubic interpolations in the first direction in order to obtain the two nearest intermediate points (S_1, S_2) and a cubic interpolation in the second direction. The two cubic interpolations applied in the first direction provide only two intermediate points, but four intermediate points are required in order to perform a cubic interpolation in the second direction. Thus, we propose an algorithm that provides better quality than the NN and less hardware resource consumption than the linear one. In order to enhance the quality of the

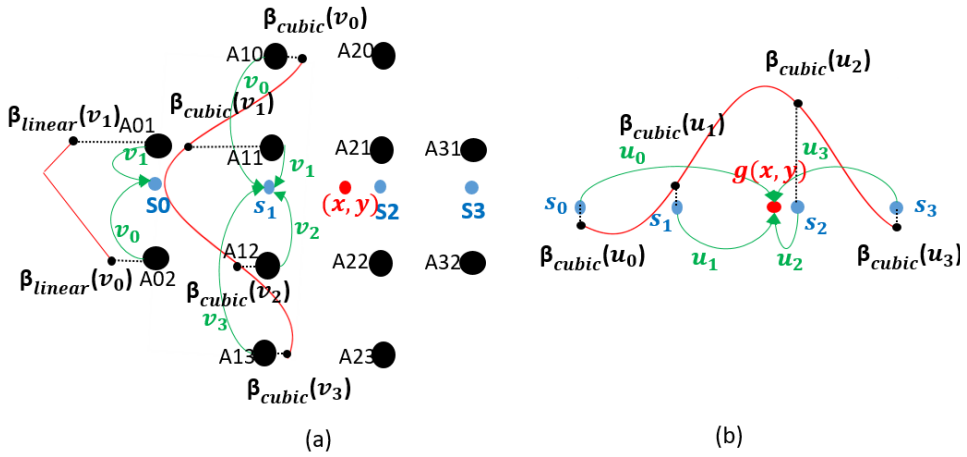

Fig. 10. Combination of linear and cubic interpolations. (a) Two cubic and two linear interpolations. (b) Cubic interpolation.

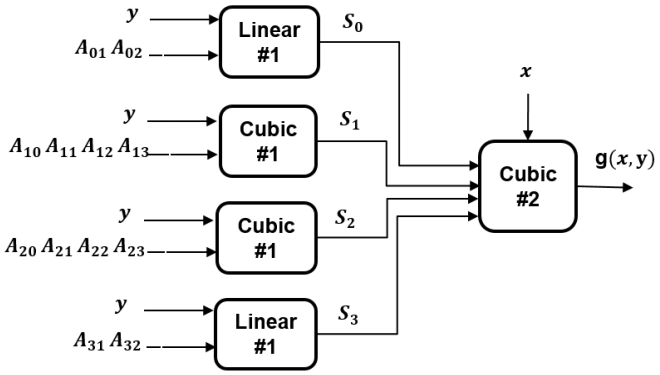

Fig. 11. Block diagram of three cubic and two linear interpolations.

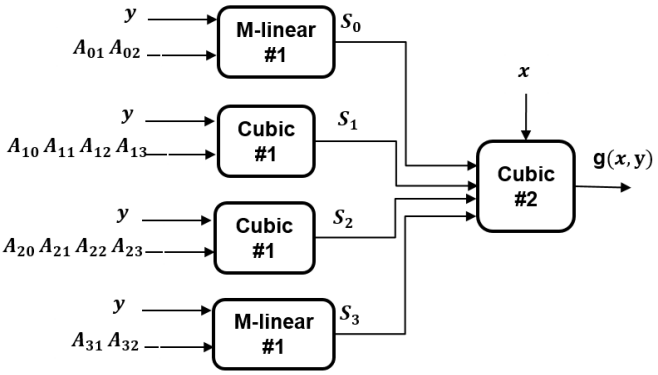

Fig. 13. Block diagram of three cubic and two modified linear interpolations.

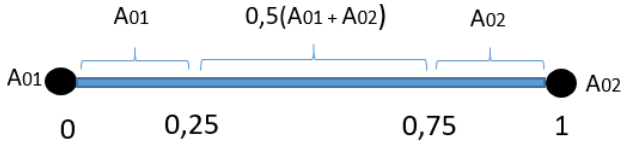

Fig. 12. Modified linear interpolation strategy.

interpolation, we proposed to split interval [0, 1] into three subintervals [0, 0.25], [0.25, 0.75], and [0.75, 1]. Concerning subintervals [0, 0.25] and [0.75, 1], the NN algorithm is applied, and for subinterval [0.25, 0.75], a mean of the two neighbors is applied, as shown in Fig. 12.

The architecture of this algorithm is shown in Fig. 13. The values S_0 and S_3 are obtained based on the idea proposed earlier. Compared with the previous algorithm, the hardware resource consumption of this algorithm is reduced (gain of two multipliers and two adders).

The same maximum isotropy error as the previous algorithm is obtained because "$\beta_{\text{mlinear}}(y_1) + \beta_{\text{mlinear}}(y_2) = 1$" is also verified.

The error due to the hardware representation (bit width) is given in Table VIII.

The global error for this architecture is equal to the error of the last block, which is $E_{\text{inter}} = E_{\text{Cubic\#2}} = 21 \times 2^{m+1-k}$.

TABLE VIII
INTERPOLATION ERROR VERSUS MULTIPLIER SIZE (k) AND PIXEL DATA LENGTH (m)

Block	Maximum error
M_linear#1	0
Cubic#1	$7 \times 2^{m+1-k}$
Cubic#2	$\mathbf{21 \times 2^{m+1-k}}$

The requirements of the proposed algorithms earlier are summarized in Table IX.

VI. RESULTS

This section provides and details the implementation results of all the 2-D algorithms described earlier. A study of their interpolation quality is also given.

For each algorithm, hardware resource consumption and maximum operating frequency are given for an Intel Cyclove V target. A classic block made with a shift buffer that receives serial pixels stream from the camera and extracts the neighbor pixels required for the interpolation is used.

In addition to the proposed algorithms, simplified architectures proposed in [21] and [23] are also implemented on the same hardware target to provide a fair comparison.

TABLE IX

REQUIREMENTS OF THE PROPOSED ALGORITHMS. IN THE MEMORY BUFFER COLUMN, THE NUMBER IN BRACKETS REPRESENTS THE NUMBER OF LINES REQUIRED WHEN APPLYING HORIZONTAL THAN VERTICAL INTERPOLATIONS

Algorithm	Memory access	Memory Buffer	Adders	Multip	Maximum $E_{Isotropy}$	Hardware precision
1cubic_4linear	8 reads	1 (3) lines	21	7	0.125×2^m	$15 \times 2^{m+1-k}$
2cubic_1linear	8 reads	3 (1) lines	28	7	0.125×2^m	$16 \times 2^{m+1-k}$
3cubic_2linear	12 reads	3 (3) lines	43	11	0.015×2^m	$25 \times 2^{m+1-k}$
3cubic_2mlinear	12 reads	3 (3) lines	41	9	0.015×2^m	$21 \times 2^{m+1-k}$

TABLE X

IMPLEMENTATIONS OF THE 3CUBIC-2MODIFIED LINEAR ALGORITHM USING 9 AND 18 FRACTIONAL BIT WIDTHS

Fractional bit-width	Multipliers (size)	LUTs	Registers	ALMs	Frequency	Maximum rounding error
9	9 (18×18)	2112	1671	1057	212 MHz	4.10E-2
18	9 (27×27)	2832	2301	1417	197 MHz	8.01E-5

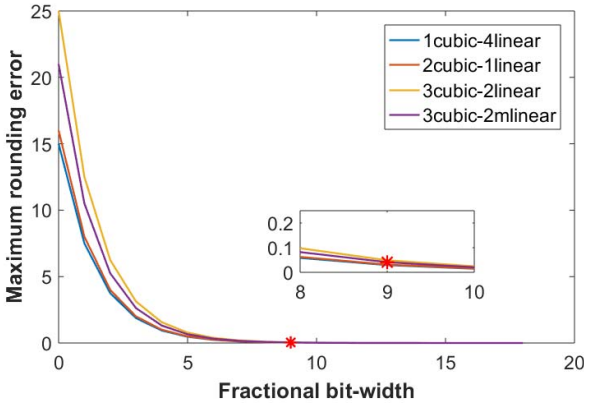

Fig. 14. Maximum rounding error in gray level versus fractional bit width.

The architecture proposed in [22] was not implemented because of its higher hardware complexity compared to [21] and [23]. All these algorithms are coded in VHDL and implemented on an Intel Cyclove V target.

The effect of the bit width on image quality and hardware resource consumption has been evaluated in order to select this parameter for the hardware implementation. It was then possible to define the best tradeoff between two main factors.

First, image quality was addressed by representing the maximum rounding error according to the variation of fractional bit width, as shown in Fig. 14. This is based on the study of the precision (or rounding error) versus the bit width reported in Table IX by considering that the pixel data length m is equal to 8.

The hardware resource consumption was then examined. As in FPGAs, multipliers of size equal to 9×9, 18×18, and 27×27 are generally provided, and two versions of one of the proposed algorithms, namely the 3cubic-2modified linear algorithm, were implemented. The first version is based on 18 bits and the second one is based on 27 bits. The results obtained in these cases and their maximum rounding errors are given in Table X.

It can be seen in Fig. 14 and Table X that the best choice is nine fractional bits, which corresponds to 18×18 multipliers. This leads to nearly the same results as those obtained in the case of floating point and reduces the hardware cost by more than 25% compared to the implementation with 18 fractional bits. Consequently, nine fractional bits were considered in this study.

The results of implementations (post place and route) obtained from Quartus for each algorithm are shown in Table XI. It is worth noting that the present results refer to a single interpolation core in a streaming application, which means that the operating frequency in megahertz implies the throughput in megapixel per second.

In order to compare the approximation quality of each method or the error generated by each algorithm compared to the bicubic one, the mean square error (MSE) is provided in the following. The MSE of an image $N \times M$ is calculated as defined in (21), where $P(i, j)$ and $P'(i, j)$ are the reference and the approximated pixel values, at the location (*i,j*), respectively. The reference value is the value obtained by using the bicubic interpolation and the approximated value is obtained by the proposed algorithms

$$\text{MSE} = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} [P(i, j) - P'(i, j)]^2. \tag{21}$$

The approximation error of each algorithm changes from one location to another. Thus, we should cover almost all possible locations. The interpolation quality is therefore evaluated by performing an image interpolation at all locations (x, y), where x and y take the values from 0 to 1 with a step of "0.1." Seven test images (Bridge, Baboon, Boat, Lake, Lena, Peppers, and Fingerprint) of size 512×512 were used. The average MSE value of each algorithm is given in Table XI.

In addition, the peak signal-to-noise ratio (PSNR) is used to measure the quality of the interpolation of each algorithm. The seven images used in the MSE calculation are downscaled and

TABLE XI
HARDWARE IMPLEMENTATIONS BASED ON AN INTEL CYCLONE V TARGET

Architecture	Multipliers	Adders	Neighbor pixels	LUTs	Registers	ALMs	Frequency	MSE
[21]	32	31	16	3078	2233	1543	158 MHz	Reference
[22]	36	40	16	na	na	na	na	Reference
[23]	20	15	16	870	574	514	209 MHz	427
2-piecewise	5	25	16	902	688	452	213 MHz	24.73
4-piecewise	20	39	16	2512	1879	1277	175 MHz	12.01
6-piecewise	20	39	16	2730	1959	1399	154 MHz	1.66
1cubic-4linear	7	21	8	1800	1586	901	210 MHz	3.99
2cubic-1linear	7	28	8	1357	1131	679	205 MHz	3.99
3cubic-2linear	11	43	12	2470	2046	1236	212 MHz	0.03
3cubic-2mlinear	9	41	12	2112	1671	1057	212 MHz	0.08

TABLE XII
PSNRS OF THE PROPOSED ALGORITHMS AND THE BICUBIC ONE WITH A SCALING-FACTOR OF 3 (PSNR/NORMALIZED PSNR TO BICUBIC). THE VALUES ARE ROUNDED TO THREE DECIMAL DIGITS

Images	Lena	Bridge	Baboon	Boat	Lake	Peppers	Fingerprint
Bi-cubic	30.112 / 1.000	22.684 / 1.000	21.828 / 1.000	25.090 / 1.000	27.642 / 1.000	29.497 / 1.000	26.155 / 1.000
2-piecewise	26.482 / 0.879	22.228 / 0.980	21.378 / 0.979	23.912 / 0.953	25.242 / 0.913	26.022 / 0.882	23.307 / 0.891
4-piecewise	29.464 / 0.978	22.589 / 0.996	21.677 / 0.993	24.885 / 0.992	27.204 / 0.984	28.820 / 0.977	25.901 / 0.990
6-piecewise	30.096 / 0.999	22.672 / 0.999	21.824 / 1.000	25.066 / 0.999	27.641 / 1.000	29.506 / 1.000	26.197 / 1.002
1cubic-4linear	30.053 / 0.998	22.785 / 1.004	22.106 / 1.013	25.099 / 1.000	27.703 / 1.002	29.669 / 1.006	25.121 / 0.960
2cubic-1linear	29.958 / 0.995	22.838 / 1.007	22.004 / 1.008	25.189 / 1.004	27.714 / 1.003	29.608 / 1.004	25.477 / 0.974
3cubic-2linear	30.109 / 1.000	22.685 / 1.000	21.826 / 1.000	25.092 / 1.000	27.642 / 1.000	29.501 / 1.000	26.159 / 1.000
3cubic-2mlinear	30.108 / 1.000	22.682 / 1.000	21.825 / 1.000	25.090 / 1.000	27.640 / 1.000	29.501 / 1.000	26.143 / 1.000

TABLE XIII
PSNRS OF THE PROPOSED ALGORITHMS AND THE BICUBIC ONE WITH A SCALING-FACTOR OF 9/4 (PSNR/NORMALIZED PSNR TO BICUBIC). THE VALUES ARE ROUNDED TO THREE DECIMAL DIGITS

Images	Lena	Bridge	Baboon	Boat	Lake	Peppers	Fingerprint
Bi-cubic	32.984 / 1.000	25.019 / 1.000	23.770 / 1.000	27.561 / 1.000	29.898 / 1.000	31.293 / 1.000	30.015 / 1.000
2-piecewise	22.907 / 0.694	21.926 / 0.876	20.447 / 0.860	22.392 / 0.812	22.231 / 0.744	22.175 / 0.709	21.562 / 0.718
4-piecewise	26.168 / 0.793	23.525 / 0.940	21.979 / 0.925	24.748 / 0.898	25.067 / 0.838	25.212 / 0.806	25.241 / 0.841
6-piecewise	29.995 / 0.909	24.564 / 0.982	23.214 / 0.977	26.645 / 0.967	28.049 / 0.938	28.752 / 0.919	28.295 / 0.943
1cubic-4linear	32.753 / 0.993	24.975 / 0.998	23.990 / 1.009	27.440 / 0.996	29.935 / 1.001	31.501 / 1.007	28.271 / 0.942
2cubic-1linear	32.540 / 0.987	25.081 / 1.002	23.913 / 1.006	27.466 / 0.997	29.932 / 1.001	31.397 / 1.003	28.846 / 0.961
3cubic-2linear	33.056 / 1.002	25.032 / 1.001	23.766 / 1.000	27.595 / 1.001	29.931 / 1.001	31.379 / 1.003	30.062 / 1.002
3cubic-2mlinear	33.055 / 1.002	25.023 / 1.000	23.762 / 1.000	27.591 / 1.001	29.920 / 1.001	31.378 / 1.003	30.060 / 1.001

then upscaled by a factor 3 and 9/4. The same interpolation method as the one used in upscaling was considered for the interpolation during downscaling. The PSNRs obtained in each case are given in Tables XII and XIII.

The maximum rounding error (in the worst case) that can occur by using nine fractional-bits hardware implementation is 0.04 gray level (see Table IX and Fig. 14), which is very small. Furthermore, the results obtained when downscaling and then upscaling the seven benchmark images by using nine fractional bits feature the same PSNR as that obtained by using the floating point. Thus, it can be concluded that using nine fractional bits do not impact the interpolation quality.

The two-piecewise linear functions technique is the simplest one with only 5 multipliers, 25 adders, and 452 ALMs. It is also more accurate than [23], which requires 20 multipliers and 514 ALMs and features an MSE value equal to 427. However, it significantly reduces the interpolation quality of the bicubic algorithm and features the lowest interpolation quality compared with the other proposed algorithms (see PSNRs in Tables XII and XIII).

The cubic-4linear and 2cubic-linear algorithms provide a similar MSE value equal to 3.99, but 2cubic-linear is advantageous in terms of hardware resource cost. Both require also the same number of multipliers and neighbor pixels (eight pixels) to perform the 2-D interpolation. These two algorithms require less hardware resources compared with [21]–[23] and provide a good approximation of the bicubic interpolation.

The four-piecewise linear functions approximation enhances the interpolation quality (decreases the MSE to 12.01) compared with two-piecewise linear functions. However, it

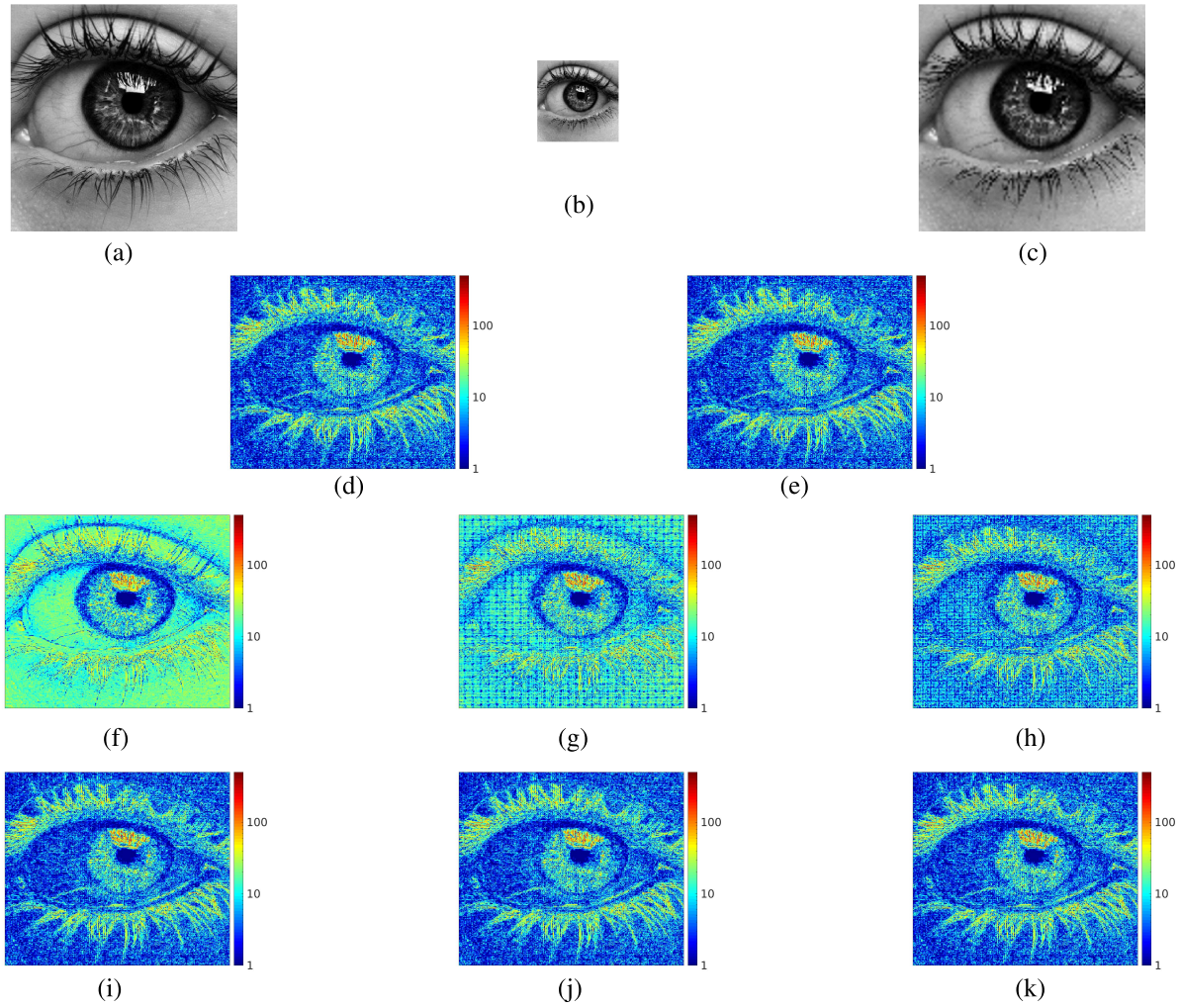

Fig. 15. Obtained results of downscaling then upscaling a reference image by a factor of 3.25 using the proposed algorithms. (d)–(k) Absolute errors compared to reference image in log scale. (a) Reference image. (b) Reference image after downscaling. (c) After bicubic upscaling. (d) Bicubic, PSNR = 24.057. (e) 3cubic-2modified-linear, PSNR = 24.055. (f) Two-piecewise, PSNR = 21.426. (g) Four-piecewise, PSNR = 22.501. (h) Six-piecewise, PSNR = 23.533. (i) Cubic-4linear, PSNR = 24.284. (j) 2cubic-linear, PSNR = 24.295. (k) 3cubic-2linear, PSNR = 24.055.

represents a lower quality (higher MSE and lower PSNRs) and higher hardware resource consumption compared with the cubic-4linear and 2cubic-linear algorithms.

The six-piecewise linear functions algorithm reduces the MSE to 1.66, which means a better quality of approximation compared with cubic-4linear and 2cubic-linear algorithms. It requires less hardware resources compared with [21] and [22], almost the same hardware resource consumption compared with four-piecewise linear functions, but 50% more than the cubic-4linear and 2cubic-linear algorithms.

The 3cubic-2linear algorithm is the best one in terms of approximation quality with an MSE value equal to 0.03. It requires lower hardware resources compared with six-piecewise linear functions [21], [22]. Furthermore, it requires only 11 multipliers instead of 20, 32, and 36 as in [21]–[23].

The 3cubic-2modified linear algorithm keeps almost the same accuracy (MSE = 0.08) as the 3cubic-2linear algorithm with lower design cost. It reduces the hardware cost by more than 20% compared to the 3cubic-2linear algorithm. As shown in Tables XII and XIII, it is clear that the 3cubic-2modified linear and 3cubic-2linear algorithms provide similar PSNRs as the bicubic ones.

Fig. 15 shows the results obtained by downscaling and then upscaling a reference image of size 377×377 by a factor of 3.25. The proposed algorithms and the bicubic one are used. The absolute errors of the obtained images compared with the reference one are given in order to better visualize and compare the interpolation quality. We can see that the proposed algorithms based on combining linear and cubic interpolations present similar quality (or error) as the bicubic one. n-piecewise linear functions algorithms present higher error, especially in the cases $n = 2$ and $n = 4$.

In FPGAs, multipliers are precious, and reducing their utilization means decreasing the hardware cost. As shown in Table XI, our proposed algorithms based on combining linear and cubic interpolations significantly reduce the multipliers use. Compared to [21] and [22], the number of multipliers is reduced by more than 65%. Furthermore, these proposed

algorithms require less ALMs and registers, operate at a higher frequency compared to [21], and approximate at best the bicubic interpolation.

The choice of one of these algorithms relies on the target application requirements. For instance, in the case of metrological application [25], the 3cubic-2modified linear algorithm is an efficient alternative to the standard bicubic interpolation.

VII. Conclusion

In this work, we studied a set of algorithms proposed as alternatives to bicubic interpolation. These algorithms are based on combining cubic and linear interpolations. The architecture of each algorithm is implemented on an Intel Cyclove V target. Then, the consumption of the hardware resource is studied. The quality of the proposed algorithms is evaluated by comparing their MSE. The main advantage of the proposed algorithms is the reduction of hardware resource consumption.

References

[1] T. M. Lehmann, C. Gonner, and K. Spitzer, "Survey: Interpolation methods in medical image processing," *IEEE Trans. Med. Imag.*, vol. 18, no. 11, pp. 1049–1075, Nov. 1999.

[2] L. Roszkowiak, A. Korzynska, J. Zak, D. Pijanowska, Z. Swiderska-Chadaj, and T. Markiewicz, "Survey: Interpolation methods for whole slide image processing," *J. Microsc.*, vol. 265, no. 2, pp. 148–158, Feb. 2017.

[3] M.-S. Pan, X.-L. Yang, and J.-T. Tang, "Research on interpolation methods in medical image processing," *J. Med. Syst.*, vol. 36, no. 2, pp. 777–807, Apr. 2012.

[4] C.-H. Kim, S.-M. Seong, J.-A. Lee, and L.-S. Kim, "Winscale: An image-scaling algorithm using an area pixel model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 6, pp. 549–553, Jun. 2003.

[5] E. Aho, J. Vanne, K. Kuusilinna, and T. D. Hamalainen, "Comments on 'winscale: An image-scaling algorithm using an area pixel model,'" *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 3, pp. 454–455, Mar. 2005.

[6] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-29, no. 6, pp. 1153–1160, Dec. 1981.

[7] C. E. Duchon, "Lanczos filtering in one and two dimensions," *J. Appl. Meteorol.*, vol. 18, no. 8, pp. 1016–1022, Aug. 1979.

[8] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *Computer Vision*. Cham, Switzerland: Springer, 2014, pp. 184–199.

[9] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 295–307, Feb. 2016.

[10] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1637–1645.

[11] W. Shi *et al.*, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1874–1883.

[12] T. Manabe, Y. Shibata, and K. Oguri, "FPGA implementation of a real-time super-resolution system using flips and an RNS-based CNN," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E101.A, no. 12, pp. 2280–2289, Dec. 2018.

[13] Z. Xiang, X. Zou, and Z. Liu, "An high quality image scaling engine for large-scale LCD," in *Proc. 8th Int. Conf. Signal Process.*, 2006, pp. 1–5.

[14] N. V. Hung, N. T. Thu Hien, P. T. Vinh, N. T. Thao, and N. T. Dzung, "An utilization of edge detection in a modified bicubic interpolation used for frame enhancement in a camera-based traffic monitoring," in *Proc. Int. Conf. Inf. Commun. (ICIC)*, Jun. 2017, pp. 316–319.

[15] W. Ruangsang and S. Aramvith, "Efficient super-resolution algorithm using overlapping bicubic interpolation," in *Proc. IEEE 6th Global Conf. Consum. Electron. (GCCE)*, Oct. 2017, pp. 1–2.

[16] C.-C. Lin, M.-H. Sheu, H.-K. Chiang, C. Liaw, and Z.-C. Wu, "The efficient VLSI design of BI-CUBIC convolution interpolation for digital image processing," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2008, pp. 480–483.

[17] C. C. Lin, M. H. Sheu, H. K. Chiang, C. Liaw, Z. C. Wu, and W. K. Tsai, "An efficient architecture of extended linear interpolation for image processing," *J. Inf. Sci. Eng.*, vol. 26, no. 2, pp. 631–648, 2010.

[18] C.-C. Lin, M.-H. Sheu, C. Liaw, and H.-K. Chiang, "Fast first-order polynomials convolution interpolation for real-time digital image reconstruction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 9, pp. 1260–1264, Sep. 2010.

[19] X. Wang, Y. Ding, M.-Y. Liu, and X.-L. Yan, "Efficient implementation of a cubic-convolution based image scaling engine," *J. Zhejiang Univ. Sci. C*, vol. 12, no. 9, pp. 743–753, Sep. 2011.

[20] P. N. Gour, S. Narumanchi, S. Saurav, and S. Singh, "Hardware accelerator for real-time image resizing," in *Proc. 18th Int. Symp. VLSI Design Test*, Jul. 2014, pp. 1–6.

[21] M. A. Nuno-Maganda and M. O. Arias-Estrada, "Real-time FPGA-based architecture for bicubic interpolation: An application for digital image scaling," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, 2005, pp. 1–8.

[22] D. Q. Liu, G. Q. Zhou, X. Zhou, C. Y. Li, and F. Wang, "FPGA-based on-board cubic convolution interpolation for spaceborne georeferencing," *ISPRS-Int. Arch. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. XLII-3/W10, pp. 349–356, Jul. 2020.

[23] Y. Zhang, Y. Li, J. Zhen, J. Li, and R. Xie, "The hardware realization of the bicubic interpolation enlargement algorithm based on FPGA," in *Proc. 3rd Int. Symp. Inf. Process.*, Oct. 2010, pp. 277–281.

[24] S. Boukhtache, B. Blaysat, M. Grédiac, and F. Berry, "FPGA-based architecture for bi-cubic interpolation: the best trade-off between precision and hardware resource consumption," *J. Real-Time Image Process.*, 2020, doi: 10.1007/s11554-020-01035-1.

[25] M. A. Sutton, J. J. Orteu, and H. W. Schreier, *Image Correlation for Shape, Motion and Deformation Measurements: Basic Concepts, Theory and Applications*. New York, NY, USA: Springer-Verlag, 2009.