A Novel Algorithm to Optimize Image Scaling in OpenCV: Emulation of Floating Point Arithmetic in Bilinear Interpolation

Samyuktha H. Subramanian

***Abstract*—Scaling is a complex algorithm wherein interpolation techniques are designed to achieve an optimum between factors such as efficiency and smoothness. One of these techniques is bilinear interpolation wherein the surrounding 2-by-2 source pixel values can be linearly weighed according to how close they are to the destination pixel. However, for applications in embedded systems with weak or no FPUs (floating point processors), the floating point values that arise after calculating weighted averages of the new pixel values have to be emulated. The algorithm presented to bypass floating point usage has been constructed based on the concept of resampling. It manipulates the pixels such that the fractional parts in the resulting floating numbers obtained are discrete in step, so that an optimal integer may be used to convert the floating point values to unsigned integers or signed integers by up-sampling. It has primarily been designed for scale factors, of integers which are multiples of 4 as well as the case of 2. This algorithm has been implemented in code in OpenCV and has been found to be faster and more efficient than the resize functions in OpenCV, and is therefore an alternative method to the scaling functions of the aforementioned library.**

***Index Terms*—Image scaling, interpolation, resampling, opencv**

I. Introduction

OpenCV, the open source computer vision library developed by Intel, contains over 500 functions[1]. One of these is the scaling function, which is equivalent to resizing an image or multiplying the matrix by a scalar. Scaling must evaluate both issues of efficiency and smoothness. While expanding an image, it is not possible to discover any more information in the image than already exists. However, there are several interpolation methods in practice such as the nearest neighbour algorithm, bilinear interpolation, bicubic interpolation which levels the original appearance after increasing the number of pixels[1].

OpenCV implements image scaling techniques in two of its library functions: *cvResize* and *resize*[4]. While the function *cvResize* takes in input arguments as the source image, the destination image and the interpolation technique, the function *resize* takes in the same input arguments along with the scaling factors for the x and y axes. These factors are double variables. The default interpolation method used is Bilinear Interpolation.

Manuscript received April 10, 2012; revised June 6, 2012.

S. H. Subramanian is with Birla Institute of Science and Technology, Pilani, India.

Implementation of these OpenCV functions in a system; which may either not have a floating point processor or have a very weak floating point processor, necessitates the elimination of floating point variables in the scaling code. The code has to be adapted for use in a fixed point processor. In this paper, an alternative code in OpenCV is presented which is based on a modified algorithm in use so as to eliminate the use of floating point variables. By using fixed point arithmetic, these programs use less space and run faster than floating-point version.

The source code of both functions shows that the primary sources of float variables in the function are from the scaling factors and the weighted averages used in Bilinear Interpolation. The algorithm has been designed for scale factors which are multiples of 4, as well as the basic case of 2. The algorithm manipulates consequent peripheral rows and columns of pixels and then uses integral up-sampling factors to bypass the declaration any floating point variables in the function. Also, none of the OpenCV structures and functions called in the code used have floating point variables defined therefore conclusively making the process accurate in its purpose.

II. Interpolation

Interpolation is the process of estimating the values of a continuous function from discrete samples[6]. Image processing applications of interpolation include image magnification or reduction, subpixel image registration, image decompression, as well as others. Of the many image interpolation techniques available, nearest neighbour algorithm, bilinear interpolation, cubic convolution interpolation and hq2x algorithms are the most common.

A. Bilinear Interpolation

Bilinear Interpolation determines the pixel intensity value from the weighted average of the four closest pixels to the specified input coordinates, and assigns that value to the output coordinates. First, linear interpolation is performed in one direction and then one more linear interpolation is performed in the perpendicular direction.

For linear interpolation, a single interpolation kernel is:

$$u(s) = \begin{cases} 0 & \text{for } |s| > 1 \\ 1 - |s| & \text{for } |s| < 1 \end{cases}$$

where s is the distance between the point to be interpolated and the grid point being considered[3].

If the weights $W, H, 1-W$ *and* $1-H$ represent the position of the destination pixel with respect to the source pixel, then Y is the new value of the destination pixel:

$$Y(J,K) = (1-W)(1-H)X(A,B) + (W)(1-H)X(A+1,B) + (1-W)(H)X(A,B+1) + (W)(H)X(A+1,B+1)$$

Equation 1

III. OpenCV Implementation Of Scaling

The scaling functions in the OpenCV library are classified as functions pertaining to Geometrical Transformations. There are two functions used in the implementation of image scaling.

The *resize* function: The return type of this function is void, and the input arguments to this function are the source image (Matrix), the destination image, the size of the destination image, the x and y scaling factors (in double variables) and the interpolation technique (which is an integer variable)

The *cvResize* function: The return type of this function is void, and the input arguments to this function are the same as the *resize* function, without the input arguments of the source and destination images.

The default scaling algorithm used is Bilinear Interpolation, and for the transformation from 8U to 32S. *In order to convert the floating point values to fixed point arithmetic, direct truncation is used by OpenCV*. This results in sizeable loss of detail while scaling. The algorithm proposed not only retains all information in the image, but also works out faster since it doesn't use any floating point libraries.

IV. Modified Algorithm

A. Concept

If two consecutive pixels, with intensity values 0 and 1, were to be scaled by a factor n, this would be the ensuing one-dimensional array of pixels formed.

1	2	.	.	.	.	.	.	.	.	2n-1	2n

Therefore, on expanding the image, by linear interpolation, each of the $2n-2$ pixels so generated will decrease from 1 to 0 in steps of $\frac{1}{2n}$

These calculations become considerably more complex in case of bilinear interpolation, where there are m and n pixels in the horizontal and vertical axes respectively. In such a case the floating point steps are in the form $1/m \times n$. In order to eliminate the usage of floating point variables, two solutions come forth. The first is truncation. The second solution would be multiplication of all these floating numbers by this factor $\boldsymbol{m \times n}$[8], however in most images; $m \times n$ is of data type long integer. Multiplying an intensity value between 0 and 255 by a long integer would result in very large numbers sparsely distributed, thus destroying continuity in intensity. For example, two consecutive pixels, with minimal difference in intensity, if multiplied by a long integer results in two numbers separated in intensity by the value of the long integer. This results in a jagged image.

In the modified algorithm, the following factors should be taken into account:

- Calculations not involving integers greater than 2^{32} (assuming a 32-bit processor)
- The up-scaling factor should be involved in the calculation of the weighted averages, and cannot be an arbitrary number. Such an arbitrary common factor would create substantial differences in intensity.
- The up-scaling factor should not be too large to cause distortion in the final image.

B. Model

In the explanation of the modified algorithm, each image is modelled as a matrix. Consider two images in the form of matrices of order $m \times n$

C. Algorithm

Consider a case where the scale factor is 2 for each dimension of the matrix.

The first step of the algorithm is filling these empty matrices with values from the original matrix that has to be scaled. Depending on the scale factor, a fraction of the scaled matrix is filled with original values. Since the scale factor is 2, half the values of each dimension (therefore a ¼ of total values of the matrix) are filled with original pixel values.

Four consecutive rows of the matrix can be visualised as:

A		C			D		B

1) Temporary Nearest Neighbour Interpolation:

All images are assumed to be in the 8U format (Bit depth: 8, type: unsigned integer)The first step of this algorithm involves temporarily writing the scaled matrix values with the original matrix values (Similar to Nearest Neighbour Interpolation) Since only a quarter of values are filled, each value will have to replicated 4 times[7]. The number of duplications depends on the scale factor. For a general $4x$ scaling factor, the number of duplications will be $16x$

The rows of pixels are now

A	A	C	C	D	D	B	B
A	A	C	C	D	D	B	B

For all forthcoming calculations, the matrix should be visualized in the form of peripheral borders such as:

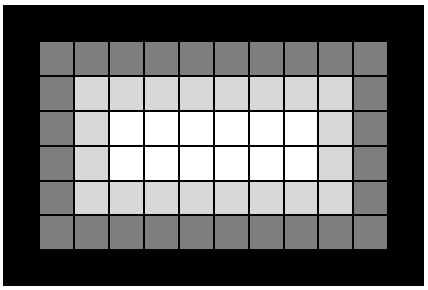

Fig. 1.

Each pixel on the outer edge( ) has 2 neighbours, A

and B.

A	O	B

Fig. 2.

The pixel value O is calculated by an averaging $\frac{A+B}{2}$ formula.

	A	
C	O	D
	B	

Fig. 3.

Each pixel on the inner borders ( ) has 4 neighbours A, B, C and D.

The value of the intensity of these pixels will be calculated by Bilinear Interpolation.

2) Edge Interpolation

The cells marked i) have 2 neighbours, and therefore, their values are calculated with the aforementioned averaging formula.

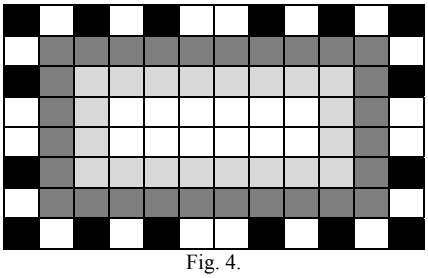

Fig. 4.

3) Bilinear Interpolation

This step calculates the intensity values of pixels which have 4 neighbours by bilinear interpolation

By method of bilinear interpolation, the formula adopted for the calculation of the intensity of each pixel is

$$(A + B + C + D)/_4$$

(Equation 1)

A, B, C and D are the pixel values as mentioned in Fig. 3.

For an arbitrary pixel O, C and D lie in the same border. (Indicated by the identical colours in Figure 1) B lies in an inner border, while A lies in the immediately outer border. The order of calculation of pixel values is from the outer periphery, via the successive borders to the centre of the matrix. Therefore, pixel values A and C have been bilinearly interpolated prior to the calculation of pixel value O, while B and D are the pixel values calculated in Figure 4. In the case of corner cells of an arbitrary border, A, C and D have been bilinearly interpolated prior to the calculation of the pixel value in the corner cell. This implies, prior to the calculation of pixel values of a particular border, pixel values of this border and the subsequently inner ones must be stored in a temporary matrix to retain the values calculated in Step I.

Consider a pixel on the outer periphery with neighbouring pixel values A and B.

The following possibilities may occur:

- A and B are odd integers, and therefore, the average of A and B is an integer.
- A and B are even integers, and therefore, the average of A and B is an integer.
- One of A and B is odd, and the other is even, in such a scenario the average is a floating point number (an integer with the addition of a fractional half)

Now, consider a pixel on the border immediately contained by the outer periphery. With respect to the 4 neighbouring pixels, the following possibilities may occur:

- A, B, C and D are integers.
- A and B are integers, while C and D are floating point numbers calculated by the edge interpolation method.
- A, B and C are integers while D is a floating point number calculated by the edge interpolation method.

If the first situation occurs, consider $\sum_1^4 N$ where N is the pixel neighbour of an arbitrary pixel O:

- If it is an integer of type $4m$, the resultant pixel value is an integer.
- If it is an integer of type $2m$, the resultant pixel value will be a floating point number of type: Integer + Fractional half (0.5)
- If it is neither of type $4m$ *or* $2m$, the resultant pixel value will be a floating point number of type: Integer + Fractional quarter. (0.25)

In the event of either the second situation occurring, it is observed that the fractional parts of the floating point number are successive multiples of the quarter. (0.25)

In the event of either the third situation occurring, it is observed that the fractional parts of the floating point number are successive multiples of the eighth (0.125)

Similarly, for the border, immediately contained by the second, the fractional parts of the floating point number are successive multiples of $1/_{32}$ (0.03125)

Therefore, with every contained border, the least count of the fractional part decreases as $1/_{2^{2n+1}}$

4) Up-Scaling

As is evident, the integer that should be multiplied by the aforementioned generated floating point numbers is 2^{2n+1}.

Analysis of the number n for both images leads to the following conclusions:

The number of consecutive borders created for a general $4m \times 4n$ is $m/_2$, where m is the lesser of the two dimensions

However, the product of the up-scaling factor and the pixel intensity value cannot be greater than 2^{32}, for a 32-bit processor.

This step is crucial in the circumvention of usage of floating point variables.

5) Iteration

The up-scaling factor with each progressive border increases by a factor 2^{2n+1}, therefore, the highest factor that can be attained while staying within the constraints of an integer type is 128, where n=3. (As $128 \times 256 = 32768$ which is the largest intensity value that can be stored in a 16 S image) Therefore, the borders visualised in Figure 1, are grouped into sets of 3, and one Edge Interpolation and two sequences of bilinear interpolation are performed on each set in iterative loops.

V. Programming The Algorithm

All images are assumed to be in the 8U format (Bit depth: 8, type: unsigned integer)

The algorithm presented in this paper has been implemented in C++ and is linked to OpenCV libraries (cv.h and highgui.h). It is an alternative to the OpenCV scaling functions. The program uses a combination of OpenCV and user defined structures. It calls the following functions and structures

- OpenCV Structures: IplImage, CvSize
- User defined Structures: CvInt: Contains an array of 3 integers. This structure is used in the storage of RGB pixel values.
- OpenCV Functions: cvLoadImage, cvGet2D, cvSet2D[4]

A minor departure from the algorithm occurs when up-sampling of each pixel value is done after temporary nearest neighbour interpolation. Thus, when bilinear interpolation and edge interpolation is performed on each pixel, the resultant values are all in fixed point arithmetic.

Also, the pixels on each border are calculated in the following order: i) Upper Row ii) Lower Row iii) Left Column iv) Right Column

The pixel values of the border being calculated and the immediately inner border are stored in the user defined structure. (CvInt) This maintains equivalency in the complexity of the fractions of all pixels in a single border.

VI. Discussion

Scaling is a ubiquitous function which finds several applications in industry today. Optimizing this function for systems with real-time computing constraints is a challenge because most of these systems either have weak floating point processors or none at all. Therefore, most functions must either compromise on loss of data, like the present algorithm OpenCV uses for scaling function (in the Bilinear Interpolation mode), since it truncates the fractional part of the floating point numbers or compromise on speed, by the inclusion of floating point libraries in the system.

The algorithm proposed in this paper works *entirely* in fixed point arithmetic. It draws motivation from the concept of resampling, which is adding detail to the image after resizing. This addition of detail is done by up-sampling each pixel value by a common factor depending on its position in the destination image. The algorithm manipulates peripheral rows and columns of pixels determine the optimum up-scaling factor. This optimum scaling factor is found to be 2^{2n+1} where $n \leq 3$. Hence, the peripheral borders are divided into sets of 3, and edge interpolation and bilinear interpolation are performed in iteration. The algorithm has been designed for scale factors which are multiples of 4, as well as the basic case of 2.

Complexity analysis of the program demonstrates this program to be faster as well as efficient since no data is lost while resampling. A compact implementation of this algorithm is an efficacious alternative to the current execution of the same function in OpenCV.

References

[1] G. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly, 2008.

[2] R. C. Gonzalez and R. E.Woods, *Digital Im*age *Processing. Prentice-Hall, Inc*, 2001.

[3] K. T. Gribbon and D. G. Bailey, A Novel Approach to Real-Time Bilinear Interpolation. *Proceedings of the Second IEEE International Workshop on Electronic Design, Test and Applications*. IEEE, 2004.

[4] Intel. OpenCV Reference Manual. Intel Corporation,2006.

[5] K. Jensen and D. Anastassiou, Subpixel Edge Localization and the Interpolation of Still Images. *IEEE Transactions on Image Processing*, 1995.

[6] P. Miklos, (n.d.). Image Interpolation Techniques.

[7] Tech-Algorithm.Com. (n.d.). Nearest-Neighbour-Image-Scaling.

[8] P. Thevenaz, T. Blu, and M. Unser, (n.d.) Image Interpolation and Resampling.

[9] M. Unser, A. Aldroubi, and M. Eden, Enlargement or reduction of digital images with minimum loss of information. *IEEE Transactions on Image Processing*, pp. 247-258,1995.

Samyuktha H. Subramanian is from Thane, Maharashtra, India. Ms. Subramanian is currently in her final semester pursuing a Bachelor's degree in Electronics and Instrumentation Engineering at the Birla Institute of Technology and Science, Pilani, India. She is currently pursuing an internship with Philips Research Asia – Bangalore, and has interned previously with the Indian Institute of Science, Bangalore. Her current research interests include the application of digital signal processing to the health care sector. She has also worked on music and speech processing prior to this.