Algorithms for the resizing of binary and grayscale images using a logical transform

Ethan E. Danahy*[a], Sos S. Agaian[b], Karen A. Panetta[a]
[a]Dept. of Electrical and Computer Eng., Tufts University, 161 College Ave., Medford, MA, USA 02155;
[b]Dept. of Electrical Eng., University of Texas/San Antonio, 6900 N. Loop, San Antonio, TX, USA 78249

ABSTRACT

The resizing of data, either upscaling or downscaling based on need for increased or decreased resolution, is an important signal processing technique due to the variety of data sources and formats used in today's world. Image interpolation, the 2D variation, is commonly achieved through one of three techniques: nearest neighbor, bilinear interpolation, or bicubic interpolation. Each method comes with advantages and disadvantages and selection of the appropriate one is dependent on output and situation specifications. Presented in this paper are algorithms for the resizing of images based on the analysis of the sum of primary implicants representation of image data, as generated by a logical transform. The most basic algorithm emulates the nearest neighbor technique, while subsequent variations build on this to provide more accuracy and output comparable to the other traditional methods. Computer simulations demonstrate the effectiveness of these algorithms on binary and grayscale images.

Keywords: image resizing, image interpolation, image reconstruction, logical transform, sum of primary implicants

1. INTRODUCTION

With the recent integration of multimedia into almost every aspect of daily lives, consumers are viewing images, video, and other visual data on a wide variety of products, ranging from televisions to computer screens to a large selection of hand-held devices. The diversity among the end users' display is far beyond what content providers can predict, thus introducing the need for algorithms capable of performing corrective signal processing to properly adapt input data to the appropriate output format [1]. Further, as more and more devices become connected to the internet, additional manipulation of the data may be necessary to overcome concerns about bit-rate and bandwidth limitations during transmission.

Image interpolation, both downspampling and upsampling, is necessary when resizing the data to match either the specifics of the communication channel or the output display. While it is more efficient to transmit low-resolution versions (often combined with data compression) to the client [2], an approximation of the high-resolution original may be necessary when presenting the final visual data. Thus, the accurate resizing of image data is an essential step in many applications, ranging from several consumer products to critical functions within the medical, security, and defense sectors.

To resize data, several techniques have been developed [3][4][5][6][7][8]. The most basic method commonly used is nearest neighbor, a procedure that is not only fast but doesn't introduce any artificial data into the final output. However, despite the speed with which is can be calculated, this procedure suffers from the fact that the resulting image often contains block artifacts, which are not only very visually noticeable but typically also can drastically negatively affect error calculations used to compare methods. Two additional techniques commonly used are bilinear and bicubic interpolation [9]. These procedures examine the points surrounding the missing data to mathematically approximate the needed values. This estimation produces smoother output that is often both visually close and quantitatively a better representation of the actual data. However, this is at the expense of computational complexity (required for performing the mathematical evaluation) and often a blurring of edges and edge details [10]. In figure 1, 1D interpolation examples are shown through a series of points to demonstrate these common techniques. Note that the bilinear and bicubic interpolation for 2D images are called linear and cubic interpolation respectively for 1D data.

*edanahy@eecs.tufts.edu; phone 1 617 627 3217; fax 1 617 627 3200

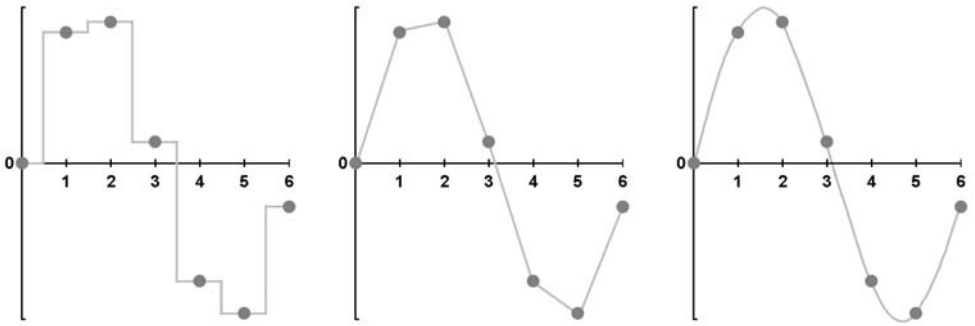


Figure 1: One-dimensional interpolation for a set of points: nearest neighbor (left), linear (middle), and cubic (right) examples.

Additional work has been done in the area of wavelet-based image interpolation [11] to try and overcome the effects of blurred edges resulting from the bilinear and bicubic methods. While success has been demonstrated by employing this technique, there are certain limitations to due the restriction that input data must be multi-bit (grayscale, for images). Thus, these methods aren't as versatile, as binary input data can not be processed as well.

This paper introduces new algorithms for the resizing of images using a logical transform. The sum of primary implicants representation is derived via a logical transform for blocks of data within the image. Analysis and manipulation of the terms found within the representation, as detailed in this paper, results in the desired scaling of the image. An initial simple algorithm duplicates the performance of the nearest neighbor method. Block mapping and edge classification result in two additional variations, improving this initial method. Initially designed for the resizing of binary data, an extension to all three algorithms introduced allow the techniques to be applied to grayscale data as well, making them functional under a wide range of input data types. Using mean squared error (MSE), the output from the algorithms can be shown to be comparable to the commonly used nearest neighbor, bilinear interpolation, and bicubic interpolation methods.

This paper is organized as follows. Section 2 details the logical transform and sum of primary implicants representation for blocks of data within the image. Section 3 presents the algorithms introduced by this work. Sections 4 and 5 provide computer simulations demonstrating the algorithms on binary and grayscale images respectively. A conclusion, found in section 6, ends the paper.

2. LOGICAL TRANSFORM

Binary data can be converted into a sum of primary implicants using a logical transform. The sum of primary implicants includes don't-care conditions for the minimized terms, which provide a more generalized representation of the data. Other logical transforms exist [12] that transform the data into minimized forms, but not into the sum of primary implicants and not always incorporating don't care conditions. Some image processing techniques employing Boolean functions [13], but not utilizing the sum of primary implicants minimized form of the data, require extensive pattern matching and other computationally intense operations due to the more complex raw data. Thus, the sum of primary implicants is incorporated here as a representation of the block data that is both minimized and includes the don't-care values [14]. Classical methods of generating this Boolean minimized form include using Boolean algebra, Karnaugh Maps, or the Quine-McCluskey method [15], all of which can be computationally intensive operations.

2.1 The logical transform

The logical system transform was introduced in [12] as a method for converting binary data into a sum of primary implicants representation. The logical transform, useful for quickly determining the Boolean minimized form (vector **y)** from an input signal **f**, is

$$\mathrm{y} = \delta_1 \left\{ \mathrm{B}_n \delta_0 \left[\mathrm{A}_n^T \delta_0 (\mathrm{f}) \right] \right\} \tag{1}$$

where δ_p is the Kronecker Delta function which accepts and returns a 1D vector where p-valued elements are set to 1 and all others to 0 and where the matrices **A** and **B** are defined as

$$\mathbf{A}_n^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}^{\otimes n}, \quad \mathbf{B}_n = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}^{\otimes n} \tag{2),(3}$$

Unlike other common transforms, the inverse operation of the logical transform is not a mirror image of the forward transform. Instead, the original binary data is generated from the sum of primary implicants through a process called implicant expansion. Equation (4) formulates one method [16] in which this can be achieved. It should be noted that the inverse operation has a much lower computational requirement than the forward logical transform.

$$\mathrm{f}_m = \sum_{t=0}^{3^n-1} \mathrm{y}_t \prod_{i=0}^{n-1} \left[\delta_{m_i}\left(\mathrm{y}_{t_i}\right) + \delta_2\left(\mathrm{y}_{t_i}\right) \right] \quad \text{for } m = 0 \text{ to } N-1 \tag{4}$$

For the previous equation, m_i is the i^{th} digit of the base-2 representation of the index m.

2.2 Logical transform examples

The following example demonstrates this process for a particular 1D input vector of length $N = 8$ (or 2^3).

$$\mathrm{f}\left(x_1, x_2, x_3\right) = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}^T \tag{5}$$

After performing the calculations in equation (1), the resulting **y**-vector (length $3^3 = 27$) has one-valued entries at indices 8 ("022" in base-3) and 21 ("210" in base-3), which translates to a sum of primary implicant representation of**:**

$$\mathrm{f}\left(x_1, x_2, x_3\right) = \text{"022"} + \text{"210"} = \overline{x}_1 + x_2\overline{x}_3 \tag{6}$$

For 2D data, such as images, blocks are scanned in a top-to-bottom, alternating left-right/right-left manner to convert them into 1D vectors for processing by the logical transform. After a gray-code reordering the indices, this vector is passed into the logical transform and again the resulting **y**-vector indicates the sum of primary implicants representation. Several examples are included in figure 2 showing binary blocks, the sum of primary implicants representation, and a count of the number of terms and number of don't care's present in each.

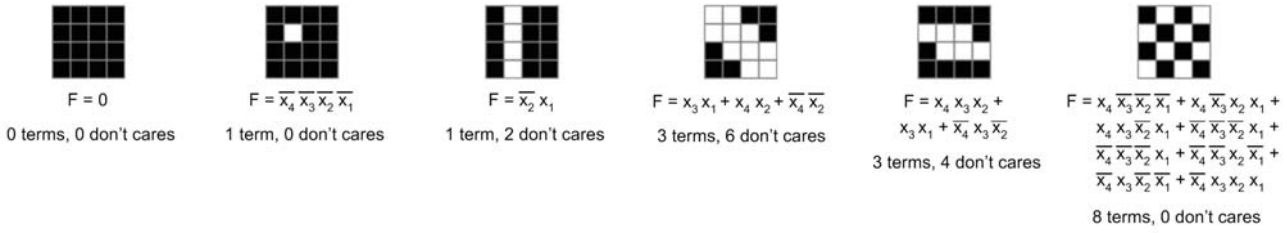


Figure 2: Sum of primary implicants representation for example 2D binary blocks.

2.3 Computational complexity

Based on fast implementations of the **A** and **B** matrix multiplication from [12] used for the calculation of the logical transform, the computational requirements for performing this operation are detailed in (7). Given N data points in the

signal, a logical transform window size of M_{LT}, and r as the number of bit planes necessary to break the original multi-bit data into binary form, the logical transform requires

$$\begin{aligned} &\frac{rN}{M_{LT}}\left(3^n - 2^n + 2n3^{n-1}\right) \\ &= \frac{rN\left(3^n - 2^n + 2n3^{n-1}\right)}{2^n} \text{ additions,} \\ &\frac{rN\left(3^{n+1}\right)}{2^n} \text{ comparisons, and } 3^n \text{ memory cells} \end{aligned} \tag{7}$$

where n is the $\log_2$ of M_{LT} (and equivalent to the number of literals in the sum of primary implicants).

3. INTERPOLATION ALGORITHMS

3.1 Algorithm 1: nearest neighbor emulation

The following figure (figure 3) shows the block representations for a 2x2 window (input) and a 4x4 window (resulting output from input being scaled up by a factor of 2). The pixel values are labeled by the variables A, B, C, and D, showing how the values are related in each window during the nearest neighbor scaling.

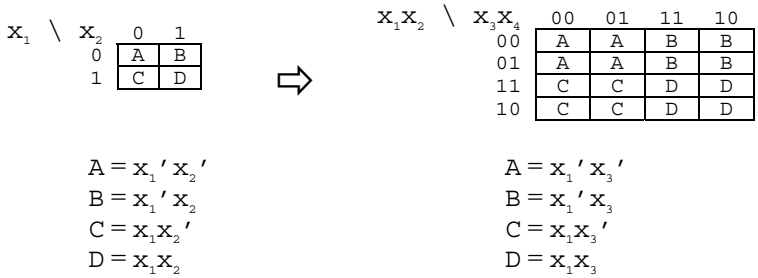


Figure 3: Nearest neighbor scaling (factor of 2) showing associated pixels from 2x2 block (left) up to 4x4 block (right).

Because the two window sizes are related, it is possible to develop a connection between the terms in the small window and the terms found in the larger nearest neighbor scaled window. For a factor of two (the same as in figure 3) and for a factor of four (2x2 window scaled to 8x8 window), this association is shown in the following diagram.

Scale factor: 2			**Scale factor: 4**		
Input window: 2x2		Output window: 4x4	Input window: 2x2		Output window: 8x8
			x_1	⇨	x_1
x_1	⇨	x_1	[don't care value]	⇨	x_2
[don't care value]	⇨	x_2	[don't care value]	⇨	x_3
x_2	⇨	x_3	x_2	⇨	x_4
[don't care value]	⇨	x_4	[don't care value]	⇨	x_5
			[don't care value]	⇨	x_6

Figure 4: Association between literals in terms of input window to literals in terms of output window.

Based on this relationship, the algorithm for performing nearest neighbor emulation using the logical transform is presented in figure 5. Due to the simplicity of the association previously shown, during implementation the procedure can be achieved either through term manipulation (software), via look-up table entries (software or hardware), or by direct literal linking/connecting (hardware). Note that since this algorithm emulates nearest neighbor exactly, the results (and resulting error) produced by this technique are identical to the output from resizing by nearest neighbor.

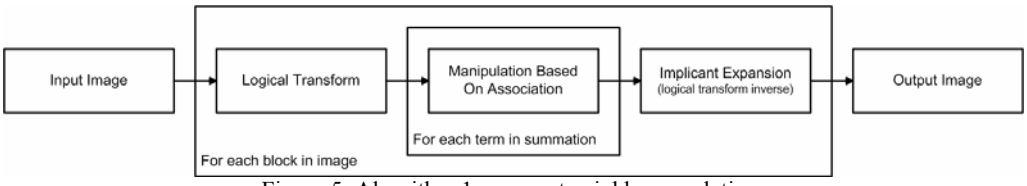


Figure 5: Algorithm 1 – nearest neighbor emulation.

3.2 Algorithm 2: term mapping

For more accurate results, a variation of this method is presented. In order to perform the resizing of the images, terms within the sum of primary implicants found within the small input data are mapped to terms within blocks of the larger, output image. This is done statistically based on a test set of representative input images. This algorithm is shown in figure 6.

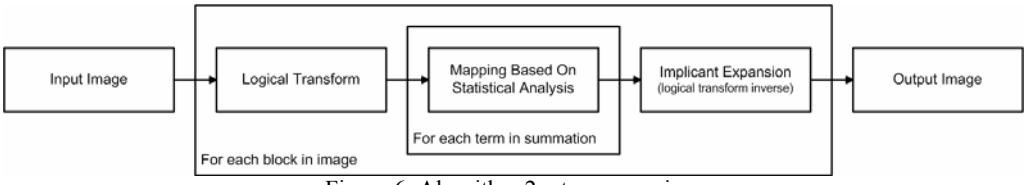


Figure 6: Algorithm 2 – term mapping.

There are two possible block orientations: direct mapping and center mapping. Direct mapping uses block sizes whose relationship is equal to the scale factor, resulting in an equivalent association between input block and output block (and results in non-overlapping block calculation during evaluation). One major disadvantage of this method is there can be negative edge-effects due to limited window size. To counter this, although at the cost of added computation, the center mapping method uses input blocks that are larger in size with only the center portion being directly associated to the output block. This alternative orientation allows more information regarding surrounding pixels to be incorporated into the term mapping analysis. However during evaluation the blocks are now overlapping, requiring additional calculations along the block edges. Figure 7 shows the relationship between these two alternatives, both for a scale factor of two (2x2 input block mapped to 4x4 output block).

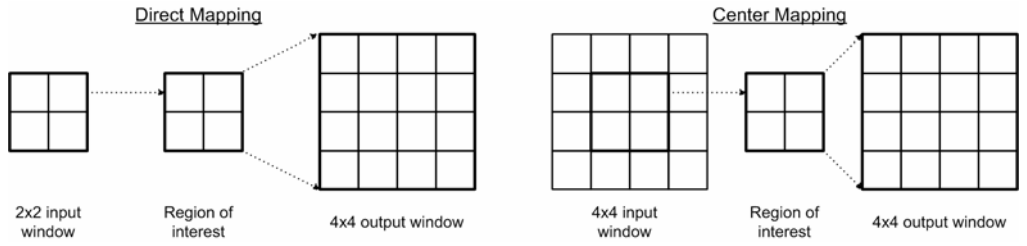


Figure 7: Comparison of direct mapping (left) to center mapping (right).

3.1 Algorithm 3: edge classification

As can be observed in the results presented later, the greatest amount of error resulting from the resizing (by the algorithms introduced in this work, as well as by those traditionally used) is found in the edge areas. For nearest neighbor, this is due to the "jagged edges" produced during scaling (see figure 10). For bilinear and bicubic interpolation, this is due to the smoothing that occurs during the approximation process.

The third algorithm variation presented here treats edge portions of the image differently than the rest of the image. This segmentation allows different approximations to be applied to the final output dependant on what portion of the image they are found. Termed "edge classification," this third variation of the algorithm performs edge detection within the image to identify and classify the edge areas of the data. This allows more precise scaling of those sections, in an

attempt to reducing blurring while still accurately approximating the correct output. Figure 8 shows this additional step in the scaling procedure.

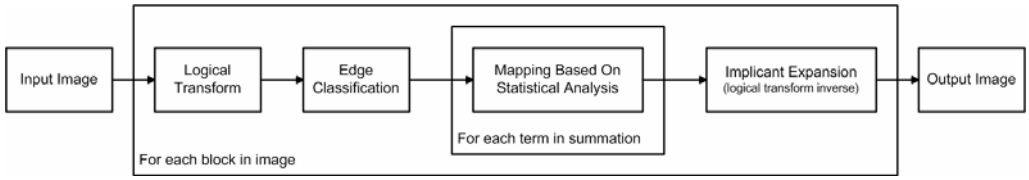
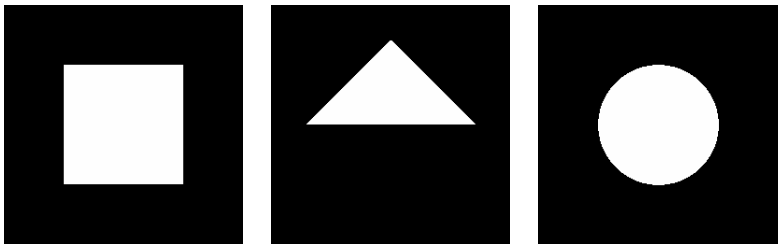


Figure 8: Algorithm 3 – edge classification.

4. COMPUTER SIMULATIONS: BINARY

To demonstrate the algorithms on binary data, a set of ten images (4 synthetic and 6 natural) of various resolutions is selected. This group is considered the ground truth to which the output of the algorithms is compared. Each initial image is scaled down to generate a set of input images to be passed into the various methods. Figure 9 shows the ten original images. Below the images are individual monikers used to reference each and the initial resolution in pixels.



Square (64x64) Triangle (64x64) Circle (64x64)

Shapes (256x256) LenaSmall (128x128) Splash (256x256) Peppers (256x256)

Barbara (512x512) Lena (512x512) Mandrill (512x512)

Figure 9: Synthetic and natural binary images used in computer simulations. The size resolution for each is shown in pixels.

The first algorithm presented here, nearest neighbor emulation, duplicates the output of nearest neighbor exactly. Therefore, visual and numerical results associated with that method are not included. For the second algorithm, term mapping, both direct mapping and center mapping windows were used, with the more accurate output selected for comparison. Finally, the edge classification in the third algorithm was achieved though use of the Canny edge detector [17][18], producing a binary edge map identifying edge locations as an intermediate step in the resizing process. Table 1 contains the mean squared error (MSE) values from the final images scaled with algorithms 2 and 3 (term mapping and edge classification respectively) as compared to the bilinear and bicubic interpolation methods.

Table 1. Binary resizing results (scale 2) comparing output MSE for various interpolation techniques

Resizing Method	Square	Triangle	Circle	Shapes	LenaSm	Splash	Peppers	Barbara	Lena	Mandrill
Bilinear Interpolation	0.0041	0.0029	0.0026	0.0118	0.0404	0.0137	0.0296	0.0724	0.0289	0.1312
Bicubic Interpolation	0.0041	0.0029	0.0026	0.0118	0.0403	0.0137	0.0296	0.0727	0.0289	0.1312
Algo 2: term mapping	0.0000	0.0015	0.0025	0.0115	0.0382	0.0136	0.0326	0.0715	0.0281	0.1350
Algo 3: edge classification	0.0000	0.0007	0.0024	0.0109	0.0379	0.0126	0.0278	0.0683	0.0280	0.1289

For binary images resized at scale 2, the outputs of bilinear and bicubic are almost equivalent. Algorithm 2 (term mapping) outperforms the two traditional interpolation techniques in almost every case. Algorithm 3 (edge classification) succeeds in equaling or improving the algorithm 2 results for every image included here. Of particular note: the square image, where the edges aligned along the window boundaries perfectly, was reconstructed exactly, resulting in zero error within the output images.

Figure 10 shows details of an edge along of the triangle image. This simple example shows an enlarged area with jagged nearest neighbor output vs. the smoother result derived from algorithm 3 (edge classification).

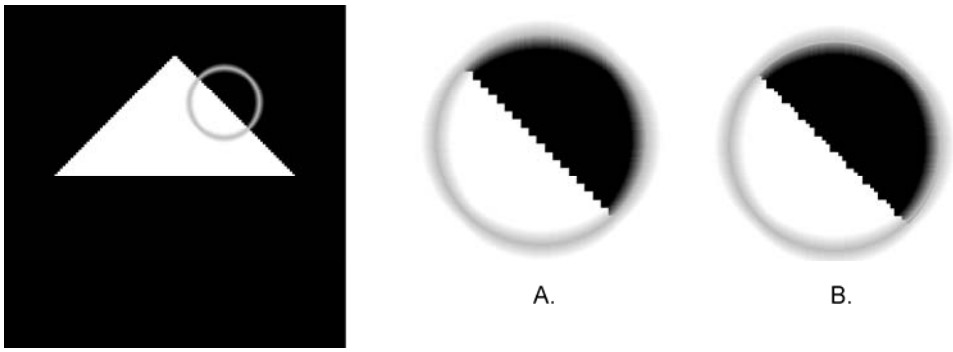


Figure 10: Triangle image (left) with highlighted area; enlarged versions show output from (A) nearest neighbor and (B) algorithm 3.

Figure 11 provides additional details from the Mandrill binary image. Close-up inspection of the eye portion of this image highlights the scaling differences produced by bicubic interpolation compared to the output from algorithm 2 (term mapping).

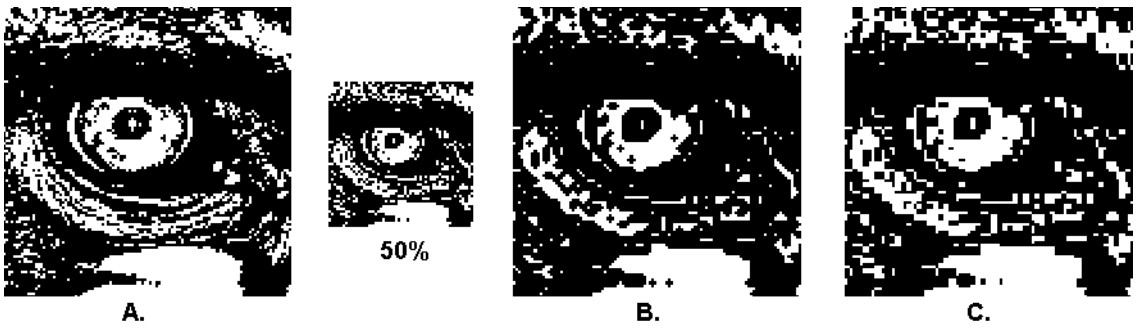


Figure 11: (A) Close-up of Mandrill eye and enlargements by (B) bicubic interpolation and (C) algorithm 2 (term mapping)

5. COMPUTER SIMULATIONS: GRAYSCALE

The algorithms introduced in this work were initially designed for binary data. However, an extension to grayscale is possible, allowing the same technique to be applied to a wide range of input data. Through bit plane decomposition, a grayscale image is broken into a series of binary data sets. Each set is then processed independently, and reassembled back into the final image at the end of the process. Figure 12 depicts this generalized procedure used in each instance.

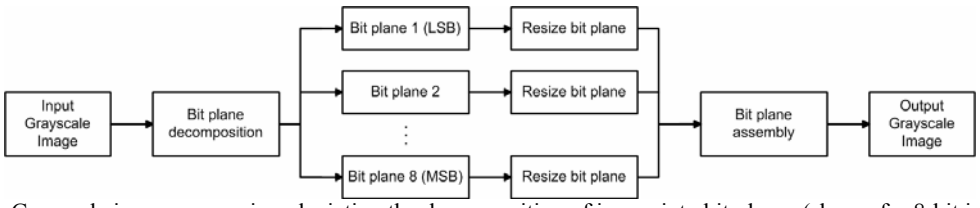
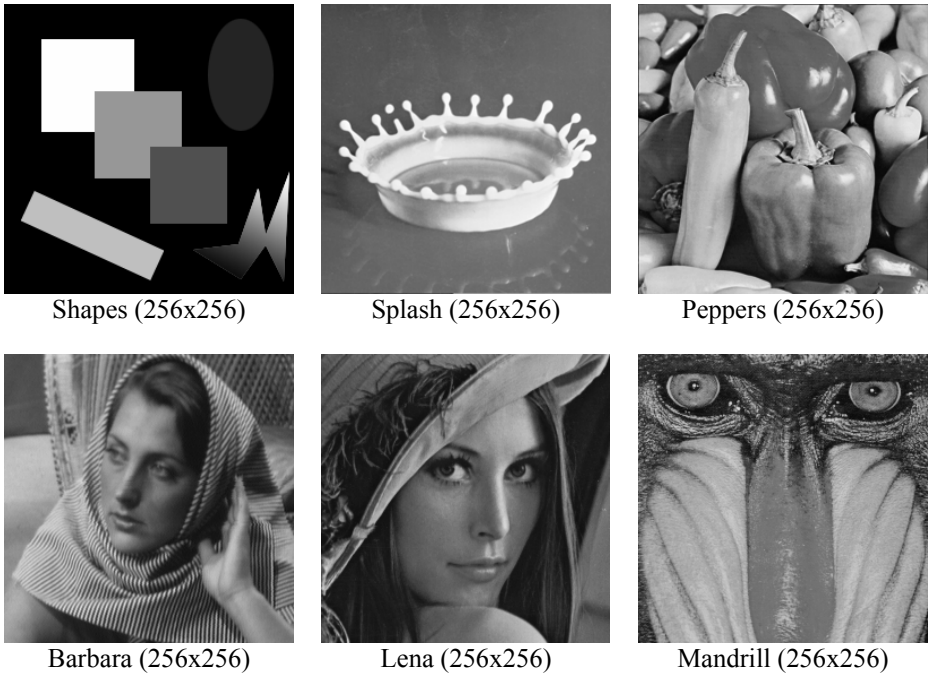


Figure 12: Grayscale image processing, depicting the decomposition of image into bit planes (shown for 8-bit input data).

Six grayscale images are selected to demonstrate the algorithms. This collection is shown in figure 13.



Shapes (256x256) Splash (256x256) Peppers (256x256)

Barbara (256x256) Lena (256x256) Mandrill (256x256)

Figure 13: Synthetic and natural grayscale images used in computer simulations. The size resolution for each is shown in pixels.

As mentioned before, the results for the first algorithm (nearest neighbor emulation) are not shown, as those duplicate the output of the nearest neighbor technique exactly. Algorithms 2 and 3 (term mapping and edge classification respectively) are performed as described previously, with the MSE output values shown in table 2 as compared to bilinear and bicubic interpolation.

Table 2. Grayscale resizing results (scale 2) comparing output MSE for various interpolation techniques

Resizing Method	Shapes	Splash	Peppers	Barbara	Lena	Mandrill
Bilinear Interpolation	0.00201	0.00035	0.00096	0.00366	0.00038	0.00666
Bicubic Interpolation	0.00203	0.00027	0.00084	0.00411	0.00033	0.00705
Algo 2: term mapping	0.00179	0.00107	0.00135	0.00568	0.00159	0.00497
Algo 3: edge classification	0.00172	0.00091	0.00020	0.00261	0.00121	0.00243

For grayscale images, algorithm 2 (term mapping) did not outperform the traditional interpolation methods, except in the cases of the Shapes and Mandrill input images. However, when edge classification was introduced to the procedure, as in the algorithm 3 specification, the resulting output was the best in every case (except for the Lena image).

Figure 14 shows details for the sequence of the Barbara images. From the close-up inspection of the results, one can see the blurring caused by the bicubic method as compared to the sharpness maintained by the edge classification technique (algorithm 3).

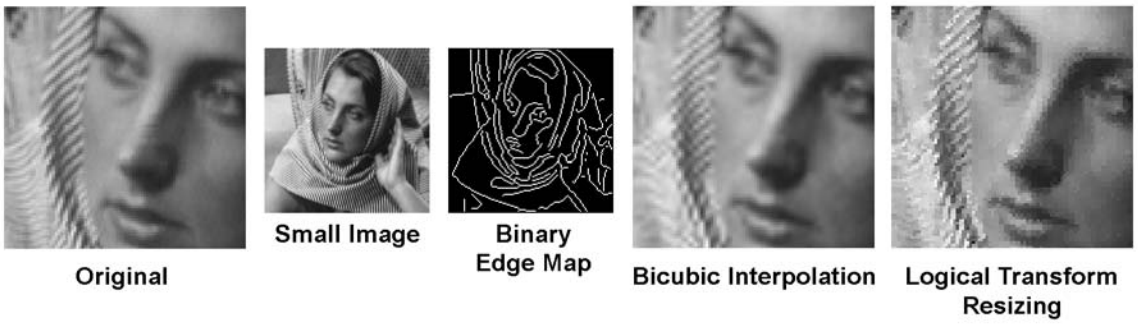


Figure 14: Barbara image showing (left to right) original, small, edge map, bicubic interpolation, and logical transform output.

6. CONCLUSION

This paper introduced three algorithms for the resizing of binary and grayscale images using a logical transform. Based on analysis of the sum of primary implicants representation of blocks within the input images, scaling techniques were applied to generate output data that is comparable to bilinear and bicubic interpolation, two commonly used methods. The first algorithm, nearest neighbor emulation, is capable of duplicating the output generated by the nearest neighbor technique. Using this as a basis, the second algorithm variation uses term mapping, where statistical analysis of a set of test images generates a mapping between input terms and output terms. Finally, the third algorithm produces further improvements through edge classification, by segmenting the data and providing more precise scaling around the edge areas in the image, thus resulting in sharper output that doesn't suffer from blurring often seen in bilinear and bicubic interpolation. Although not demonstrated here, the same concepts presented can be extended further to 1D signals (audio, for example) and to both color images and video data as well.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0306464. This research is also supported in part by the Center for Infrastructure Assurance and Security.

REFERENCES

[1] H. Park, Y. Park, and S. Oh, "L/M-Fold Image Resizing in Block-DCT Domain Using Symmetric Convolution," IEEE Transactions on Image Processing, Vol. 12, No. 9, September 2003.

[2] H. Shu and L. Chau, "And Efficient Arbitrary Downsizing Algorithm for Video Transcoding," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, No. 6, June 2004.

[3] A. Munoz, T. Blu, and M. Unser, "Least-Squares Image Resizing Using Finite Differences," IEEE Transactions on Image Processing, Vol. 10, No. 9, September 2001.

[4] C. Hentschel, "Generic Method for 2D Image Resizing with Non-Separable Filters," IEEE International Conference on Image Processing, 2004.

[5] Y. Park and H. Park, “Design and Analysis of an Image Resizing Filter in the Block-DCT Domain,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, No. 2, February 2004.

[6] J. Hwang and H. Lee, “Adaptive Image Interpolation Based on Local Gradient Features,” IEEE Signal Processing Letters, Vol. 11, No. 3, March 2004.

[7] L. Chen and K. Yap, “Regularized Interpolation Using Kronecker Product for Still Images,” IEEE International Conference on Image Processing, Vol. 2, September 2005.

[8] Y. Park and H. Park, “Arbitrary-Ratio Image Resizing Using Fast DCT of Composite Length for DCT-Based Transcoder,” IEEE Transactions on Image Processing, Vol. 15, No. 2, February 2006.

[9] T. Blu, P. Thevenaz, and M. Usner, “Linear Interpolation Revitalized,” IEEE Transactions on Image Processing, Vol. 13, No. 5, May 2004.

[10] C. Lee, M. Eden, and M. Unser, “High-Quality Image Resizing Using Oblique Projection Operators,” IEEE Transactions on Image Processing,” Vol. 7, No. 5, May 1998.

[11] S. Chang, Z. Cvetkovic, and M. Vetterli, “Locally Adaptive Wavelet-Based Image Interpolation,” IEEE Transactions on Image Processing, Vol. 15, No. 6, June 2006.

[12] S. Agaian, J. Astola, and K. Egiazarian, Binary Polynomial Transforms and Nonlinear Digital Filters, Marcel Dekker, Inc., New York, 1995.

[13] E. Danahy, S. Agaian, and K. Panetta, “Detecting Edges in Noisy Multimedia Environments,” IEEE International Symposium on Multimedia, December 2006.

[14] E. Danahy, S. Agaian, and K. Panetta, “Filtering of Impulse Noise in Digital Signals Using Logical Transform,” Visual Information Processing XIV, SPIE Defense and Security Symposium, vol. 5817, p. 188-199, March 2005.

[15] J. Wakerly, Digital Design: Principles and Practices, 3rd ed., Prentice Hall, New Jersey, 2001.

[16] S. Agaian, T. Baran, and K. Panetta, “The Application of Logical Transforms to Lossless Image Compression Using Boolean Minimization,” Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), p. 13-31, March 2003.

[17] J. Parker, Algorithms for Image Processing and Computer Visions, John Wiley & Sons, New York, 1996.

[18] P. Bao, L. Zhang, and X. Wu, “Canny Edge Detection Enhancement by Scale Multiplication,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 9, p. 1485-1490, September 2005.