Fast Bitmap Images Resizing

Lain-Jinn Hwang*, Louis R. Chow† and Chia-Ton Tan‡

Department of Computer Science and Information Engineering
Tamkang University
Tanshui, Taipei, Taiwan 251, R.O.C.
*Email: micro@mail.tku.edu.tw
†E-mail: chaory@ms.cy.gov.tw
‡E-mail: nick@mail.mine.tku.edu.tw

***Abstract*— Image resizing is an important operation in digital image processing. In this paper, we introduce an image resizing scheme that produces significantly improved quality over the pixel replication method. This algorithm is suitable for real-time image resizing applications including: Virtual Reality, Optical Character Recognition, Symbol Recognition, and Car License Plate Recognition. Although this method produced "jaggies" at the edges in the resized image, the execution time is about 22 to 40 times fast than the Windows and the Weiman scheme, respectively.**

***Index Terms*—Image Resize, Pixel Replication, Rothstein code.**

I. Introduction

Due to the advantage in digital signal processing, more and more photography data is available today in a digital format. Image resizing is an important geometrical process in digital image processing [1]. Applications of image resizing include: Virtual Reality(VR), Optical Character Recognition(OCR), Symbol Recognition(SR), and Car License Plate Recognition(CLPR), medical imaging processing [2], and other resolution conversion which requires image realization on monitors or printers.

The most common techniques to resize image include (a) interpolation and spline method [3]; (b) frequency domain method [4], [5], [6]; (c) area re-sampling [7]; and (d) pixel replication. Usually, to achieve a high quality result, image resizing normally use interpolations. For a practical usage, interpolation is time consuming and unsuitable for a lots of real-time applications, such as VR, OCR, SR and CLPR. Although pixel replication is fast and simple, its cause some aliasing. As a result, critical lines may disappear and others may be duplicated unnecessarily. In this paper, we introduce an image resizing scheme that produces significantly improved quality over pixel replication method. Its execution time is about 22 to 40 times fast than the Windows and the Weiman scheme, respectively.

The organization of this paper is as follows. Section II introduces the basic definitions of digital image representation and discusses the most popular digital image resizing methods. In Section III, we propose a fast and efficient method base on replication to resize digital images. Experimental results and conclusions are included in Section IV and V, respectively.

II. Methods to resize Digital Image

A. Basic Definitions

An digital image $f(x, y)$ is a two-dimensional array of pixel values that specifies the intensity (brightness) of a signal at a particular spatial position (x, y), as shown in Figure 1. The origin is located at the top left corner. Note that, the row indices are from 0 to $w - 1$ and the columns range from 0 to $h - 1$. For a point (x, y) of a digital image, x and y are assumed to be integers with index intervals $0 \leq x \leq w - 1$ and $0 \leq y \leq h-1$. The values of w and h specify the *image resolution.* An image pixel value $f(x, y)$ has a range of G gray levels, where $G \geq 2$. In image processing, it is practical to use the values $G > 2$ (gray value image) and $G = 2$ (*bilevel* image). The Function $f(x, y)$ represents a gray-scale pixmap of an image.

A image might be degraded during processing. Measurement of image quality can be used to assess the degree of degradation. Methods for assessing image quality can be divided into two categories: subjective and objective.

Enlargement and reduction are lossy operations. The peak signal to noise ratio (PSNR) is a well-known and objective measure of the performance of an image resizing scheme [8]. The PSNR values between an original image f and the zoom image $\hat{f}$ is defined as (1).

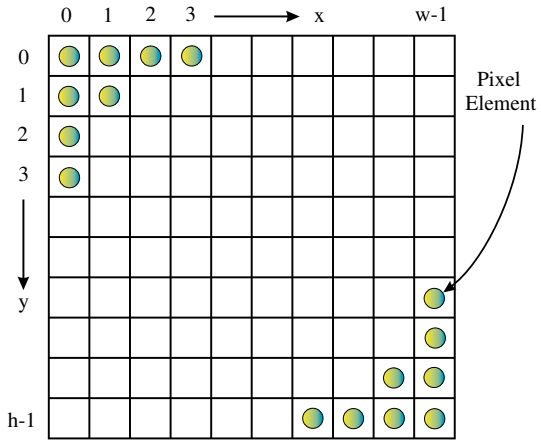


Fig. 1: Matrix representation of a digital image.

$$MSE = \frac{1}{w \times h} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} [f(x,y) - \hat{f}(x,y)]^2 \tag{1}$$

$$PSNR = 10 \log_{10} [\frac{255 \times 255}{MSE}] \text{dB}$$

B. Digital Image Resizing

There are four major approaches for digital image resizing:

1) Interpolation and Spline-based Method.
2) Frequency Domain Transformation (FFT/DCT).
3) Pixel Replication, and
4) Area Re-sampling.

1) **Interpolation and Spline-based Method.:** The most elementary type of interpolation consists of fitting a polynomial through the pixels of the image to be resized line by line. For any image line $f(\bullet, l)$, $0 \leq l \leq h-1$, we can find the $(w-1)$th Lagrange interpolating polynomial (2) to pass through the w pixels.

$$\begin{aligned} P_{w-1}(x) &= f(0,l)L_{w-1,0}(x) + \cdots + \\ &\quad f(w-1,l)L_{w-1,w-1}(x) \\ &= \sum_{i=0}^{w-1} f(i,l)L_{w-1,i}(x) \end{aligned} \tag{2}$$

where

$$L_{w-1,i}(x) = \prod_{\substack{k=0 \\ k \neq i}}^{w-1} \frac{x-k}{i-k}$$

for each $i = 0, 1, \ldots, w-1$.

One might expect that the quality of interpolation increases with an increased degree of n of the polynomials used. Unfortunately, this is not generally true. Indeed, for various functions f, the corresponding interpolation polynomials may tend to oscillate more and more between pivotal points as n increases.

An alternative approach is to divide the interval into a collection of subintervals and to construct one approximating polynomial on each subinterval. This approach is called the Piecewise Polynomial Approximation. This means that, we partition x, $0 \leqq x \leqq w-1$, into k subintervals with common endpoints, then we fit a polynomial through each subinterval. Hence, instead of approximating $f(\bullet, l)$ by a single polynomial on the entire interval $0 \leqq x \leqq w-1$, we approximate $f(\bullet, l)$ by k polynomials.

The most common piecewise polynomial approximation uses third-degree polynomial whose curve passes through four points(called cubic spline interpolation). Note that these methods, although very accurate, suffer from two drawbacks:

1) Not suitable for real time applications.(i.e., due to the complexity)
2) Not suitable for bilevel image.

2) **Fourier Transform and Cosine Transform(FT/CT):** The Fourier transform and Cosine transform play a critical role in a broad range of image processing applications, including enhancement analysis, restoration, and compression. These methods can also be used to resizing digital images [4], [5], [6]. Sid-Ahmed [4] have designed an algorithm based on 2D Fourier Transform that enlarges an image by 2. The algorithm is briefly discussed here:

1) Compute the two-dimensional Fourier Transform of a given image of size $N \times N$, i.e.,$F = FFT2D(f)$.
2) Append zeros to the F to increase its size to $2N \times 2N$.
3) Compute the two-dimensional Inverse Fourier Transform of F. The result is the original image with size $2N \times 2N$.

The main disadvantage of these methods is computationally expensive. It must perform forward two-dimensional transform and inverse two-dimensional transform.

3) **Pixel Replication:** The simplest way to magnify pixels is to duplicate pixel by an integer number in the X or the Y direction(or both). Expansion of images in integral multiples is fairly easy – just duplicate every pixel column or row N times. Shrinking of an image by a factor of $1/N$ is achieved by removing every Nth column or row. However, when we want to scale

an image by a non-integer scalar, the procedure becomes more complex. We have to replicate or remove pixel rows or columns according to a special pattern.

The basis for determining this pattern is the Rothstein code [7], a binary sequence representing a line with slope p/q. In the case of expanding an image horizontally, it tells us how to distribute q columns of the source image among p columns of the target by duplicating each appropriate column to fill in the gaps. For example, if we are expanding an image with q columns to a width of p columns, the Rothstein code will be a set of p bits with q of them set, each set bit evenly distributed among the p bits. Each set bit indicates that this column is to be copied, and each unset bit indicates that the column is to be a duplicate of the copied columns.

For reduction an image, we can use the inverse of the expansion slope, the factor q/p. The Rothstein code tells us which columns to remove. Again, each column with a marked bit in the Rothstein code is copied and each column with an unmarked bit is removed.

The generation of the Rothstein code can be described graphically by Fig 2. First, we plot a line with the desired slope on a grid. Every column in which the line crosses a horizontal grid line receives a one, all others receive a zero. Algorithm 1 shows the routine to calculate the Rothstein code. In every column we increase $xsum$ by n, which is the minimum of p and q. When $xsum$ overtakes m, which is the maximum of p and q, the line crosses a horizontal grid mark. The Rothstein code for the current column is set to one and m is subtracted from $xsum$. This process continues until the number of columns has been reached.

For reducing of an image line from 7 pixels to 5 pixels horizontally, the Rothstein code is 0110111 as shown in Fig 2. Thus, pixel 0 and 3 are removed, which generates an image of 5 pixels. To expand an image line from 5 pixels to 7 pixels, because the first digit of the Rothstein code is 0, we can rotate the code to the left by one digit to get 1101110, and use this code to generate the new image with pixel 1 and pixel 4 copied. Although this method is simple and fast, but this method will create "jaggies" at the destination image if the source image have an edge.

4) **Area re-sampling:** This method resizes images using the average of its nearest neighbors. Weiman [7] supposed an algorithm based on cyclic permutation of the Rothstein code to get

Algorithm 1 Generate of Rothstein code for p/q.

Require: $roth$: array [0..MAX(p, q)-1] of char;
{array to store Rothstein code.}
p, q : integer; {Scale factor of p/q.}
m : integer; {max of p and q.}
n : integer; {min of p and q.}
xi : integer; {array index.}
xsum : integer; {accumulator.}

Ensure: Rothstein code in array $roth$.

```
m = max(p, q);
n = min(p, q);
xsum = 0;
for xi = 0 to m − 1 do
    xsum = xsum + n;
    if xsum >= m then
        roth[xi] = 1; {set to 1.}
        xsum = xsum − m;
    else
        roth[xi] = 0; {clear to 0.}
    end if
end for
```

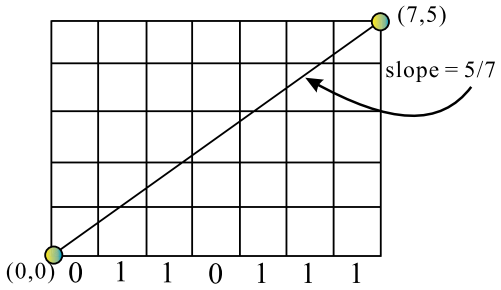


Fig. 2: Rothstein's code for a line of slope $\frac{5}{7}$.

different mappings from sources to destinations, and to add the mapped image to a temporary image. Then, the algorithm takes average of the temporary image to get a resized image. Foley [9] gives an pseudo code based on Weiman Algorithm to expand an image. Although this method overcomes the "jaggies" problem and gets a high PSNR value, but it requires additional memory buffer and summation/average operation. This drawback makes the algorithm not suitable for real time applications.

III. Image Resizing by Random Sampling

By using Rothstein code, Pixel Replication can resize an image by a non-integer scale factor. We have developed an algorithm that directly resizes an image line from width w to W, for any $w, W > 0$, as shown in Algorithm 2. This algorithm combines the enlargement and the reduction schemes in a single algorithm,

making it suitable of implementation in many programming languages.

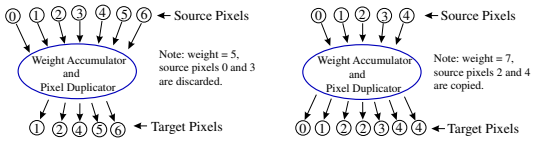
Algorithm 2 Resize of an image line from width w pixels to W pixels.

Require: source : grayscalePixmap; {Source image, size w.}
target : grayscalePixmap; {Target image, size W.}
w : integer; {width of source image, $w > 0$.}
W : integer; {width of target image, $W > 0$.}
x : integer; {Indices for source line.}
tx : integer; {Indices for target line.}
AccX : integer; {weight accumulator.}
Ensure: Resized image in target with width W.

```
1:  AccX = 0;
2:  tx = 0;
3:  for x = 0 to w − 1 do
4:      AccX = AccX + W;
5:      while AccX >= w do
6:          {copy source pixel to target.}
7:          target[tx] = source[x];
8:          tx = tx + 1; {next pixel}
9:          AccX = AccX − w;
10:     end while
11: end for
```

To resize an image line from width w to W, this algorithm sequentially adds W to $AccX$, if $AccX > w$, the algorithm copies the corresponding source pixel to a target pixel, and subtracts w from $AccX$. Otherwise, the algorithm removes the corresponding source pixel.

Fig 3 schematically represents the proposed technique. Cell contents represent pixel indices. According to this algorithm, to reduce an image line from width 7 to 5, pixels 0 and 3 are removed from the source line. To expand this image line from 5 to 7, pixels 2 and 4 are duplicated.



(a) reduce 7 pixels to 5 pixels (b) expand 5 pixels to 7 pixels

Fig. 3: Zoom by Pixel Replication.

Although this scheme is fast and simple, it may cause some aliasing. As a result, some horizontal/vertical may disappear and others may be duplicated too many times. This algorithm may causes the lose of symbol-based information in the source image.

This problem can be eliminated by generating a random number, which forces the duplicated or removed pixels randomly distributed among the pixels or lines. Algorithm 3 shows the algorithm to resize an image from $w \times h$ to $W \times H$.

This algorithm generates uniform random numbers rz and cz. Where rz is between 0 and $h-1$, which is used to randomly select a line of image for duplication or skipping. And, cz is between 0 and $w-1$, which is used to randomly select a column of image for duplication or skipping. By introducing the duplication or skipping scheme, we can prevent a horizontal/vertical line "all disappear" of the Pixel replication scheme, and improve the visual effect of the resized image.

Algorithm 3 Resize of an image from $w \times h$ to $W \times H$.

Require: source : grayscalePixmap; {Source image, size $w \times h$.}
target : grayscalePixmap; {Target image, size $W \times H$.}
w, h : integer; {Size of source image, $w > 0, h > 0$.}
W, H : integer; {Size of target image, $W > 0, H > 0$.}
x, y : integer; {Indices for source image.}
tx, ty : integer; {Indices for target image.}
rz, cz : integer; {zoom of column and row}
Ensure: Resized image in target with size $W \times H$.

```
1:  {Generate an uniform random number between 0 and h − 1.}
2:  rz = uniform(0, h − 1);
3:  ty = 0;
4:  for y = 0 to h − 1 do
5:      rz = rz + H;
6:      while rz >= h do
7:          {Generate an uniform random number between 0 and w − 1.}
8:          cz = uniform(0, w − 1);
9:          tx = 0;
10:         for x = 0 to w − 1 do
11:             cz = cz + W;
12:             while cz >= w do
13:                 {copy source pixel to target.}
14:                 target[ty][tx] = source[y][x];
15:                 tx = tx + 1; {next pixel}
16:                 cz = cz − w;
17:             end while
18:         end for
19:         ty = ty + 1;{next line of target}
20:         rz = rz − h;
21:     end while
22: end for
```

IV. **Experimental Results**

In order to illustrate the performance of Random Duplication scheme, we conduct a series of complementary image magnification and reduction test, based on four 256×256 images: Baboon, Barbara, Cameraman, and Lena. We test the three algorithms and add an off-the-shelf Microsoft Windows API function – StretchBlt, for comparison. Four approaches are used in the comparison:

1) Microsoft Windows API function - StretchBlt.
2) Weiman algorithm.
3) Pixel replication.
4) Random replication.

The experiments are implemented on a PC(Pentium II 300, running Windows 2000 with 512MB SDRAM). The execution time of the resizing schemes is too small to measure by using the general built-in library functions, such as "time", this function gets the resolution up to "second". The execution time of the various resizing schemes was measured by using the "Read Time-Stamp Counter" instruction of the Intel Architecture(which is implemented in an assembly language). The "Time-Stamp Counter" is a 64-bit counter register, which is increased by 1 of each clock cycle when the processor starts. This instruction load the processor's current 64-bit time-stamp counter into a register. For each image, we perform the resizing operation by each scheme, and measure the PSNR value, compute the CPU clock cycles of enlargement and reduction operations. This task is repeats 100 times for the sake of Random Replication. The speedup ratio is evaluated by the following equation:

$$\frac{\text{CPU clock cycles of the other schemes}}{\text{CPU clock cycles of the Random replication}}$$

In a series of two experiments, we first enlarge the image to 320×320 and then reduced to its original size of 256×256. Table I shows the PSNR values of these four schemes. The experimental results show that the proposed algorithm improves PSNR values over Pixel Replication(PR) by 4.26 dB average. Although the PSNR values are lower than the ones of StretchBlt(WIN) and Weiman(WEI) Algorithm, the execution time is about 13 to 22 and 30 to 41 times faster, respectively, as show in Table II.

In the second experiment, we first reduce the test images to 192×192, then enlarge them to their original size 256×256. Table III shows the PSNR values of the four schemes. Fig 4 illustrate the PSNRs and the execution time of the Barbara image. The resulting images of Barbara from the four methods are shown in Fig 5. The speed up times is shows in Table IV. The experimental results show that the proposed algorithm has an improved PSNR value over Pixel Replication by 2.56 dB in average. Again, the PSNR value is lower than StretchBlt and Weiman Algorithm. But its execution time is speed up about 17 to 23 and 22 to 40 times, respectively.

TABLE I: PSNR values(Enlarge to 320×320 then reduce to 256×256).

				Random		
image	WIN	WEI	PR	Min.	Avg.	Max.
Baboon	35.71	25.94	16.54	17.02	20.80	23.79
Barbara	35.67	34.92	23.54	25.49	28.31	30.97
Cman	35.77	31.94	20.18	21.85	25.03	27.85
Lena	35.71	34.84	23.10	25.51	27.91	30.41

TABLE II: Speedup times, relative to Random replication(enlarge to 320×320 then reduce to 256×256).

	Windows API – StretchBlt		Weiman	
image	Enlarge	Reduce	Enlarge	Reduce
Baboon	13.03	21.10	29.87	40.42
Barbara	13.08	21.15	29.83	40.48
Cman	13.13	21.81	29.53	40.55
Lena	13.61	21.98	29.96	40.72

TABLE III: PSNR values(Reduce to 192×192 then enlarge to 256×256).

				Random		
image	WIN	WEI	PR	Min.	Avg.	Max.
Baboon	19.96	22.35	16.54	16.84	18.19	19.31
Barbara	27.14	31.17	23.54	24.81	26.00	26.81
Cman	24.73	27.91	20.18	21.31	22.74	23.96
Lena	27.01	31.06	23.10	24.56	25.66	26.39

TABLE IV: Speedup times,relative to Random(reduce to 192×192 then enlarge to 256×256).

	Windows API – StretchBlt		Weiman	
image	Reduce	Enlarge	Reduce	Enlarge
Baboon	21.73	17.10	40.42	21.69
Barbara	22.08	17.11	33.33	21.71
Cman	22.52	17.26	33.77	21.88
Lena	22.74	17.12	33.35	21.71

V. **Conclusion**

We have presented a new image resizing scheme suitable for real-time applications such as Virtual Reality animation, Optical Character Recognition, Symbol Recognition, and Car License Plate Recognition. This algorithm is based on the Pixel replication method, which combines

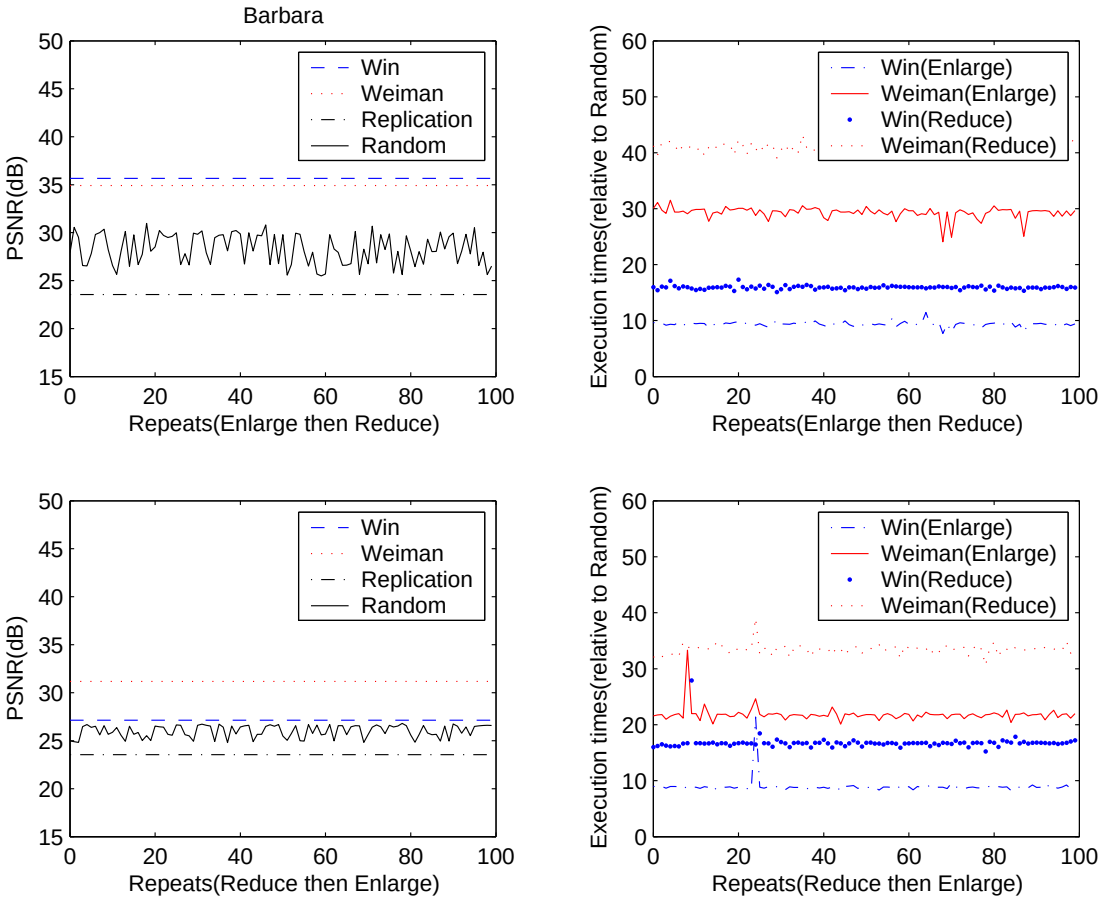


Fig. 4: PSNR and Speedup times of Barbara.

the enlargement and the reduction schemes in a single algorithm. Although this method produces "jaggies" at the edges in the resized image, however, its execution time is about 13 to 23 and 30 to 41 times faster than a Windows API function and the Weiman Algorithm, respectively.

REFERENCES

[1] W. K. Pratt, *Digital Image Processing: PIKS Inside*. New York: John Wiley and Sons, Inc., 2001.

[2] T. M. Lehmann, C. Gonner, and K. Spitzer, "Survey: Interpolation Methods in Medical Image Processing," *IEEE Trans. Med. Imag.*, vol. 18, pp. 1049–1075, Nov. 1999.

[3] M. Unser, A. Aldroubi, and M. Eden, "Fast B-Spline Transforms for Continuous Image Representation and Interpolation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, pp. 277–285, Mar. 1991.

[4] M. A. Sid-Ahmed, *Image Processing: Theory, Algorithms and Architectures*. New York: McGraw-Hill, Inc., 1994.

[5] S. A. Martucci, "Image Resizing in the Discrete Cosine Transform Domain," in *IEEE Int. Conf. Image Processing*, vol. 2, pp. 244–247, May 1995.

[6] T. C. Chen and R. J. P. D. Figueiredo, "Image Decimation and Interpolation Techniques Based on Frequency Domain Analysis," *IEEE Trans. Communications*, vol. 32, pp. 479–484, Apr. 1984.

[7] C. F. R. Weiman, "Continuous Anti-Aliased Rotation and Zoom of Raster Images," *SIGGRAPH '80 Proc.*, vol. 14, pp. 286–293, July 1980.

[8] J.-K. Han and S.-U. Baek, "Parametric Cubic Convolution Scaler for Enlargement and Reduction of Image," *IEEE Trans. Consumer Electronics*, vol. 46, pp. 247–256, May 2000.

[9] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*. Reading, Massachusetts: Addison-Wesley, 1990.

Fig. 5: Zoom of Barbara.