Halftoning Through Optimization of Restored Images — A New Approach with Hardware Acceleration

Tetsuo Asano Koji Nakano

January 22, 2003
IS-RR-2003-001

School of Information Science
Japan Advanced Institute of Science and Technology
Tatsunokuchi, Ishikawa, 923-1292 Japan

ISSN 0918-7553

Halftoning Through Optimization of Restored Images — A New Approach with Hardware Acceleration

Tetsuo Asano
School of Information Science
JAIST
t-asano@jaist.ac.jp

Koji Nakano
School of Information Science
JAIST
knakano@jaist.ac.jp

January 22, 2003

Abstract

We present a new approach to find a binary image reproducing a given continuous-tone image so that the picture printed out on a paper by a printer with limited number of ink colors looks as similar to the given one as possible. This process is called digital halftoning. The main contribution of this paper is to present a new approach for digital halftoning to generate binary images of best quality. Our basic strategy is well characterized as halftoning through optimization of restored image, that is, we minimize the difference between an input image and the image restored from an output binary image by applying a usual Gaussian filter. Algorithmically, our approach uses iterative exhaustive search on huge neighborhood combined with several acceleration techniques. The experimental results are fairly satisfactory: the resulting binary images are quite sharp and have no artifact. Since a simple iterative exhaustive search needs a lot of computing time, we have developed the FPGA-based co-processor for acceleration. Using the co-processor, artifact-free sharp binary images can be obtained in practical computing time.

1 Introduction

Digital Halftoning is an important task to convert a continuous-tone image into a binary image consisting only of black and white dots [7, 4]. This task is necessary when we print out a monochrome or color image by a printer with limited number of ink colors. A great number of techniques and algorithms for digital halftoning have been proposed so far. Among them, Error Diffusion [6] which propagates rounding errors to unprocessed neighboring pixels according to some fixed ratios is the most commonly used algorithm. It preserves the average gray level between input and output images. It is quite efficient and often gives excellent results, but one of its drawbacks is visible artifact especially in an area of uniform intensity. Several techniques have been developed to avoid creating artifacts in output images [11]. However, as long as Error Diffusion based techniques are used, the pixel values are propagated to neighbors and the resulting images are defocused.

One of our goals in this paper is to devise an algorithm that finds an artifact-free sharp binary image while keeping the similarity to an input image as much as possible. The similarity or dissimilarity between input and output images is defined using the human vision system. That is, for the purpose of comparing an input multiple-level images and its associated output binary image we restore the output image by applying a Gaussian filter. The dissimilarity between input and output images is defined by the sum of difference of gray levels in the input image and the resulting restored image at each pixel. In other words, it is the difference between an input image and an image restored from an output binary image by a Gaussian filter.

Several halftoning algorithms have been proposed in this direction. One of them is Direct Binary Search [1], which is an iterative improvement or minimization of the difference between input and output images. Some variations are possible using heuristic search methods such as simulated annealing and genetic algorithm (e.g. see [5]).

The problem of minimizing the above-defined difference between input and output images seems to be intractable. Although there is no NP-hardness result for this problem, a simpler problem is known to be NP-complete [2]. The problem is to minimize the maximum error instead of the sum of errors. The definition of error is also different: For a 2×2 small region R we take the sum of values in input and output images and their difference is defined as the error for the region. If we take all such 2×2 regions to evaluate the maximum error, the problem is NP-complete. On the other hand, if we carefully design a family of such regions (not necessarily 2×2) so that an incidence matrix defined between pixels and regions is totally unimodular (simply speaking, each pixel belongs to a few regions), then an optimal solution (optimal halftoning) is obtained in polynomial time by applying a minimum-cost network flow algorithm [3].

Our algorithm to be presented uses iterative improvement, but what should be distinguished from the existing ones is local search on huge neighborhood with hardware acceleration and several algorithmic techniques for efficient implementation. In the usual local search for halftoning used to be flipping one or two pixel values while our new algorithm repeats an operation of replacing a binary pattern for a group of pixels with the locally best binary pattern for the group. Such optimization is done using the exhaustive search with some acceleration techniques. It is also supported by hardware implementation using FPGA (Field Programmable Gate Array) technology.

We have implemented our algorithm both in a usual environment of a single CPU PC and also using FPGA as a hardware accelerator. The results are quite satisfactory. Although there is no formal proof for the optimality of our output images, the authors strongly believe that they are almost optimal and it would be impossible to distinguish them from theoretically optimal images even if we could find them.

2 Preliminary

Suppose that an original L-level gray-scale image $A = (a_{i,j})$ of size $n \times n$ is given, where $a_{i,j}$ denotes the intensity level at position (i, j) $(1 \leq i, j \leq n)$ taking an integer in the range $[0, L-1]$. The goal of halftoning is to find a binary image $B = (b_{i,j})$ of the same size that reproduces original image A, where each $b_{i,j}$ is either 0 or 1. We measure the goodness of output binary image B using the Gaussian filter that approximates the characteristic of the human visual system. Let $V = (v_{k,l})$ denote a Gaussian filter, i.e. a 2-dimensional symmetric matrix of size $(2w+1) \times (2w+1)$, where each $v_{k,l}$ $(-w \leq k, l \leq w)$ is determined by a 2-dimensional Gaussian distribution. The image $R = (r_{i,j})$ restored from a binary image $B = (b_{i,j})$ by

applying the Gaussian filter is an L-level gray-scale image:

$$r_{i,j} = \left\lfloor (L-1) \sum_{-w \le k,l \le w} v_{k,l} b_{i+k,j+l} \right\rfloor \quad (1 \le i,j \le n) \tag{1}$$

From $\sum_{-w \le k,l \le w} v_{k,l} = 1$, each $r_{i,j}$ takes an integer in the range $[0, L-1]$ and restored image R is an L-level gray-scale image. We can say that a binary image B is a good approximation of original image A if the difference between A and R is very small. According to this consideration, we are going to define the error by the difference between R (or B) and A as follows. The error $e_{i,j}$ at each pixel location (i,j) is defined by

$$e_{i,j} = a_{i,j} - r_{i,j}, \tag{2}$$

and the total error [1] is defined by

$$Error(A,B) = \sum_{1 \le i,j \le n} |e_{i,j}|. \tag{3}$$

Since the Gaussian filter approximates the characteristics of the human visual system, we can think that image B reproduces original gray-scale image A if $Error(A,B)$ is small enough. The best binary image that reproduces A is a binary image B with the minimum total error is given by the following formula:

$$B^* = \arg\min_B Error(A,B). \tag{4}$$

Once the best image B^* is obtained, the restored image R, which can be computed by (1), is a good approximation of A.

If the size of the Gaussian filter is 1×1, then B^* can be obtained by the simple thresholding. In other words, the binary image $B^* = (b_{i,j})$ such that $b_{i,j} = 1$ if and only if $a_{i,j} \ge \frac{L}{2}$ is an optimal binary image satisfying (4). However, in general, it is very hard to find the optimal binary image B^* for a given gray-scale image A if the Gaussian filter is not small, say, 5×5. As we discussed in Introduction, the problem of finding the optimal binary image B^* is so intractable that we believe it is NP-hard. We need to evaluate $Error(A,B)$ for all possible 2^{n^2} binary images B to find the best image. Clearly, this takes more than $\Omega(2^{n^2})$ computing time. Since n is usually larger than 100, this approach is not feasible. Thus, the challenge is to find, in practical computing time, a nearly optimal binary image B, whose total error is close to that of the optimal image B^*.

3 Iterative Improvement with Local Search on Huge Neighborhood

The main purpose of this section is to present our ideas to find a good binary image B whose error with respect to original gray-scale image A may not be minimum but is small. A basic idea behind our approach is *iterative improvement* with *local search on huge neighborhood.* We first show our local search on huge neighborhood, which is a key ingredients of our halftoning algorithm.

Suppose that an original image A and a temporary binary image B are given. Further, let $W(i,j)$ be a window of size $k \times k$ in B whose top-left corner is at position (i,j). Our first idea is to check all 2^{k^2} binary patterns in $W(i,j)$ and replace the current binary subimage in the window by the best binary pattern that minimizes the total error. In other words, we find a binary image B' such that

$$B' = \arg\min\{Error(A,B) \mid B \text{ and } B' \text{ differ only in } W(i,j)\}. \tag{5}$$

Clearly, $Error(A,B') \le Error(A,B)$ always holds, we can say that B' is an improvement of B as a reproduction of original gray-scale image A.

[1] We can generalize the total error using L_p metric such that $Error(A,B) = [\sum_{1 \le i,j \le n} |e_{i,j}|^p]^{1/p}$. However, this paper deals with only $p = 1$.

Next, let us see the details how we find B' satisfying formula (5) above. Since we use a Gaussian filter of size $(2w+1) \times (2w+1)$, the change of the binary pattern may affect the errors in a square region of size $(2w+k) \times (2w+k)$, which we call the *influence region*. The reader should refer to Figure 1 for illustration of a window, a Gaussian filter, and the influence region. It should be clear that the best binary pattern can be selected by computing the total errors of the influence region of size $(2w+k) \times (2w+k)$, because the change of the binary pattern does not affects errors at pixels outside the influence region.

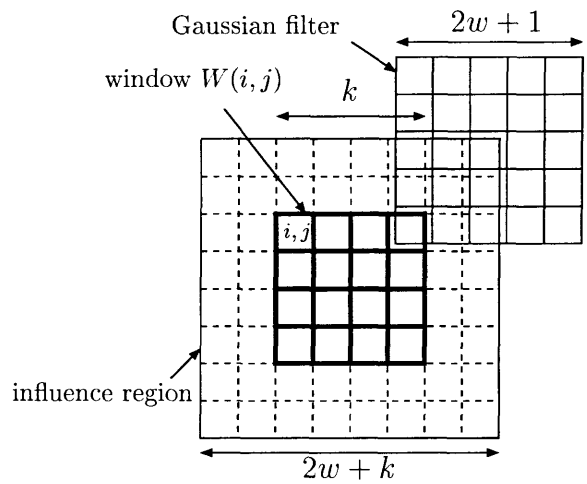

Figure 1: Illustrating a window of size $k \times k$, a Gaussian filter of size $(2w+1) \times (2w+1)$, and the influence region of size $(2w+k) \times (2w+k)$

Let us evaluate the computing time necessary to find the best binary pattern in the window. The error of a fixed pixel in an influence region can be computed in $O(k^2)$ time by evaluating formulas (1) and (2). Hence all the errors in the influence region can be computed in $O(k^2(2w+k)^2)$ time. After that, their sum can be computed in $O((2w+k)^2)$ time. Thus, the total error in the influence region can be computed in $O(k^2(2w+k)^2)$ time. Since we need check all the possible 2^{k^2} binary patterns, the best binary pattern can be obtained in $O(2^{k^2}k^2(2w+k)^2)$ time. We can improve the computing time by a pixel flipping in the order of the gray code of binary numbers. Recall that the gray code represents a list of all 2^m binary numbers with m bits such that any two adjacent numbers differ only one position. Thus, by flipping a bit in an appropriate position, we can list all the 2^m binary numbers with m bits. Using the gray code with k^2 bits, we can check all binary patterns in $O(2^{k^2}w^2)$ time as follows. Starting with the current pixel pattern in the window, we repeat flipping an appropriate pixel by the gray code. In each flipping, we compute the total error in the influence region for the current binary pattern in the window. Since a single bit flipping affects the error of $2w+1 \times 2w+1$ pixels, the total error can be computed in $O(w^2)$ time in an obvious way. Thus, the best binary pattern can be computed in $O(w^2) \times 2^{k^2} = O(2^{k^2}w^2)$ time using the exhaustive search.

We are now in position to show our iterative improvement using the exhaustive search because no more improvement for the current window is possible. Let B_0 be an appropriate initial binary image B_0. We scan the binary image by a window of size $k \times k$ and improve it by the exhaustive search, that is, replacing binary pattern in the window so that the total error is minimized. The scan by a window can be done by any order. We perform the scan on B_0 in the raster order, and obtain a better quality binary image B_1. The same procedure is repeated, that is, the scan operation is applied to B_{t-1} and obtain a better binary image B_t $(t \geq 1)$ until B_{t-1} and B_t are identical and no more improvement is possible. When computing B_t for $t \geq 2$, we do not have to perform the exhaustive search for all the windows. If the restored image of the influence region for the current window did not changed, then we can omit the exhaustive search. The details of our algorithm are spelled out as follows:

Iterative Improvement(A)
Set an appropriate initial binary image in B_0;
$B_1 \leftarrow B_0$

```
for i ← 1 to n − w + 1 do
  for j ← 1 to n − w + 1 do
    Perform the exhaustive search in W(i, j) for B_1 and update B_1
    by the best binary pattern.
t ← 2;
do {
  B_t ← B_{t−1};
  for i ← 1 to n − w + 1 do
    for j ← 1 to n − w + 1 do
      if the influence regions of W(i, j) for R_{t−1} and R_{t−2} are not
      identical then perform the exhaustive search in W(i, j) for B_t
      and update B_t by the best binary pattern.
  } until (B_t and B_{t−1} are identical)
output (B_t);
```

If $k = 1$, then all we need to do is to flip the binary value of a pixel. This idea is implemented under the name of Direct Binary Search (DBS) [1]. In DBS, a pixel is flipped if the resulting image has smaller total error. Thus, our iterative improvement method is a generalization of the DBS. Since the DBS checks only the immediate neighborhood, the binary image falls into the local minimum solution with large error. However, since we check the large neighborhood, say, 4×4, the resulting binary image is very close to the optimal one. Actually, we have developed the image halftoning system using FPGAs, which performs the iterative improvement for widows of size 3×3 and 4×4. As we are going to show later, the binary images obtained by the large neighborhood are much better in the quality than those by the DBS.

4 Hardware Acceleration for the Exhaustive Search using an FPGA co-processor

We have developed the hardware accelerator using the PCI-connected FPGA that performs the exhaustive search to find the best binary pattern in a window. This section is devoted to show the architecture of our hardware FPGA-based accelerator.

Before showing the architecture, we first show how our hardware accelerator is used by the host PC. Recall that, as is easily seen in Figure 1, $W(i, j)$ contains pixels at position $(i+i, j+j')$ for $0 \leq i', j' \leq k-1$. Also, the influence region involves pixels $(i + i, j + j')$ for $-w \leq i', j' \leq w + k - 1$. The host PC initializes all binary pixels in window $W(i, j)$ to 0, i.e. $b_{i+i,j+j'} = 0$ for all $0 \leq i', j' \leq k - 1$. After that, it computes all the errors in the influence region by (2). The host PC sends these errors to the PCI-connected FPGA. Since each error is an integer in the range $[-(L-1), L-1]$, $(2w + k)^2$ integers with $(\log L + 1)$ bits are sent through the PCI bus. Note that the number of bits sent though the PCI bus is independent of the size $n \times n$ of the image. For example, if $w = 2$, $k = 4$, and $L = 256$, then the PCI bus transfers 576 bits to find the best binary pattern. Once the PCI receives the errors in the window, it computes the best pattern by the exhaustive search.

Next, we are going to show the architecture of our hardware accelerator. Figure 2 illustrates a part of the FPGA-based hardware accelerator, which outputs the total error for every binary patterns. The binary pattern is stored in the k^2-bit counter. A combinatorial circuit is used to apply the Gaussian filter to this binary pattern stored in the counter. Note that the combinatorial circuit is very simple; it just evaluates Formula (1) for the window of size $(2w + k) \times (2w + k)$. Once the resulting image and the errors are obtained, the total error is computed by a simple combinatorial circuit, which just computes the formulas (2) and (3). The evaluation of formulas (1) and (3) involves the computation of the sum of integers, which can be performed quite efficiently on the FPGA [9, 8]. The architecture illustrated in Figure 2 outputs the errors of all binary patterns in 2^{k^2} clock cycles. Using all the total errors, the best binary pattern can be stored into a register using a comparator in an obvious way. After that, the best binary pattern is sent to the host PC, which stores it in the window of the binary image.

In principle, the hardware accelerator computes the best binary pattern in 2^{k^2} clock cycles. Since the circuit is so simple that it applies the Gaussian filter and computes the total sum, we can use the pipelining

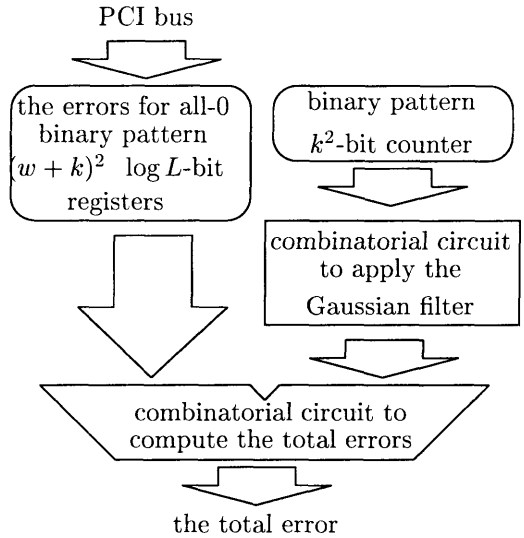

Figure 2: Illustrating a part of the hardware accelerator.

technique [9] to increase the frequency of clock cycles. Actually, we have partitioned the combinatorial circuits into seven stages. The pipelined architecture works correctly in frequency of 80MHz on Xilinx XC2V3000 (Speed Grade 4). We can also achieve further speed up by parallelizing the exhaustive search. Recall that the hardware accelerator is used to compute the total errors for a set of all 2^{k^2} binary patterns. We can equally partition this set into m groups of $\frac{2^{k^2}}{m}$ in obvious way and use m-parallel circuit to find the best binary patten in each group. Actually, we succeeded in embedding three 2-parallel hardware accelerators in XC2V3000, each of which computes the best binary pattern for each of the RGB colors.

5 Experimental Results

This section presents experimental results obtained by our halftoning method using iterative improvement on huge neighborhood.

We have developed a software that performs our halftoning algorithm using the iterative improvement for windows of sizes 1×1, 2×2, 3×3, and 4×4, which we call *1-flip*, *4-flip*, *9-flip*, and *16-flip*, respectively. Recall that 1-flip corresponds to the DBS [1].

Table 1 show the results for a standard color image, "Lena", which is a 256-Level color RGB image of size 512×512. We use a Gaussian Filter of size 5×5 with parameter $\sigma = 1.5$. The iterative improvement is applied to each of the RGB colors independently. The initial binary image is given by the white noise halftoning, which gives 1 to each pixel $b_{i,j}$ with probability $\frac{a_{i,j}}{L}$.

The Average Error in Table 1 is the absolute value of the error per pixel, that is, $(Error(A_r, B_r) + Error(A_g, B_g) + Error(A_b, B_b))/3n^2$, where $A_r, A_g, A_b, B_r, B_g, and B_b$ denote the images of RGB colors of original A and halftone image B, respectively. Clearly, the average error for 16-flip is the smallest and that for 1-flip (DBS) is quite large. We have used a Pentium4 Processor PC (2.8GHz, 2Gbyte memory, 512Kbyte cache) with Linux OS (kernel 2.4). Table 1 includes the computing time by the software on this PC using no FPGA. The source C program is complied by gcc 3.2 with -O2 and -march=pentium4 options. For "Lena", the non-FPGA program runs in 536.8 and 72834 seconds for 9-flip and 16-flip, respectively. To accelerate the computation for 9-flip and 16-flip, we use Xtreme DSP kit [10], which is a PCI card with Xilinx VirtexII series FPGA XC2V3000. Since we embedded three 2-parallel circuits on the FPGA, the exhaustive search for a window can be done for each of the RGB colors concurrently in $\frac{2^{k^2}}{2}$ clock cycles. Although the speed up factor of 97.3 is attained for 16-flip, that for 9-flip is only 4.52. The reason is that the latency of data transmission through PCI bus is dominant in computing time for 9-flip, because the exhaustive

	1-flip	4-flip	9-flip	16-flip
Average Error	7.81	5.32	4.91	4.61
Time (no-FPGA,sec)	1.91	20.56	536.8	72834
Time (FPGA,sec)	-	-	118.8	748.17
Speed up	-	-	4.52	97.3

Table 1: The experimental results for "Lena", 256-Level RGB Color image of size 512 × 512.

search for a window takes only $\frac{2^9}{2} = 256$ clock cycles, which is approximately $(\frac{256}{80MHz})sec \approx 3.2\mu sec$. On the other hand, the exhaustive search for 16-flip needs $\frac{2^{16}}{2} = 32768$ clock cycles, which is approximately $(\frac{32768}{80MHz})sec \approx 0.4msec$. If the FPGA has an enough memory to store the whole image data, we can omit the frequent data transmission though PCI bus. If this is the case, we can attain the same speed up factor as that of 16-flip and the computing time for 9-flip will be few seconds.

Figure 3 shows the resulting halftone images and their restored images including the images generated by ordered dither and Error Diffusion. It is easily to see that the quality of the binary image obtained using 16-flip is the best. The quality of 9-flip is very close to 16-flip. The binary images obtained by Error Diffusion has artifacts and worms, while those of 9-flip and 16-flip are artifact-free. Also, the restored images of 9-flip and 16-flip are quite sharp and very well reproduces the original image. We can conclude that, since the 16-flip takes a lot of time, the 9-flip is a practical method for find halftone images of good quality.

6 Discussions and Future Works

We have demonstrated a powerful halftoning algorithm for producing plausible halftone images without any visible artifacts under the ideas of iterative improvement based on local search on huge neighborhood together with the help of hardware acceleration. Since this halftoning process requires huge amount of computation, it would be impractical if we would implement it naively. Although the processing time is still much larger than that of the currently used halftoning algorithms such as Error Diffusion and also much larger hardware resources are required, our algorithm would be useful if we really needed good halftone images.

The scheme described in this paper could be applied to several other optimization problems related to halftoning. One of them is to design optimal masks (or matrices) used for blue noise mask and dot diffusion.

Acknowledgments

This work has been partially supported by Grant in Aid for Scientific Research of the Ministry of Education, Culture, Sports, Science and Technology of Japan. The authors would like to thank Hisao Tamaki for his stimulus suggestions on local search on huge neighborhood.

References

[1] M. Analoui and J.P. Allebach. Model-based halftoning by direct binary search. In *Proc. SPIE/IS&T Symposium on Electronic Imaging Science and Technology*, volume 1666, pages 96–108, 1992.

[2] T. Asano, T. Matsui, and T. Tokuyama. Optimal roundings of sequences and matrices. *Nordic Journal of Computing*, 7(3):241–256, 2000.

[3] T. Asano, K. Obokata, N. Katoh, and T. Tokuyama. Matrix rounding under the l_p-discrepancy measure and its application to digital halftoning. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 896–904, 2002.

[4] Tetsuo Asano. Digital halftoning: Algorithm engineering challenges. *IEICE Transactions on Information and Systems*, February 2003.

[5] C.H. Chu. A visual mode-based halftone pattern design algorithm. In *IEEE Signal Processing Society Seventh Workshop on Multidimensional Signal Processing*, pages 23–25, 1991.

[6] R. W. Floyd and L. Steinberg. An adaptive algorithm for spatial gray scale. *SID 75 Digest,Society for Information Display*, pages 36–37, 1975.

[7] Daniel L. Lau. *Modern Digital Halftoning.* Marcel Dekker, 2001.

[8] Rong Lin, Koji Nakano, Stephan Olariu, M. C. Pinotti, James L. Schwing, and Albert Y. Zomaya. Scalable hardware-algorithms for binary prefix sums. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):838–850, 8 2000.

[9] Koji Nakano and Koichi Wada. Integer summing algorithms on reconfigurable meshes. *Theoretical Computer Science*, pages 57–77, January 1995.

[10] Nallatech. *Xtreme DSP Development Kit User Guide*, 2002.

[11] Victor Ostromoukhov. A simple and efficient error-diffusion algorithm. In *Proc.of the 28th SIGGRAPH*, pages 567 – 572, 2001.

Figure 3: The resulting halftone images and the restored images: **Top row**, Ordered Dither, Error Diffusion(Floyd and Steinberg), and the halftone image by 1-flip(DBS). **Second row**, the halftone images by 4-flip, 9-flip, and 16-flip. **Third row**, Original image, the restored image of Error Diffusion, and that of 1-flip. **Bottom row**, the restored images of 4-lip, 9-flip, and 16-flip.