# Fast error diffusion and digital halftoning algorithms using look-up tables

*Chai Wah Wu; IBM T. J. Watson Research Center; Yorktown Heights, NY, Mikel Stanich, Hong Li, Yue Qiao, Larry Ernst; IBM Printing Systems Division; Boulder, CO*

## Abstract

*Recently, a fast error diffusion halftoning algorithm using look-up tables (LUT) was proposed to speed up the multiplication of error filter coefficients. In this letter, we propose another LUT-based error diffusion halftoning algorithm which is more flexible in terms of the size of the LUT that can be used and thus allows for a more optimal tradeoff between halftone quality, processing speed, hardware complexity and parallelizability. Furthermore, the aggregate error in the proposed algorithm can be computed with different bitdepths for the different errors. As an example, we present a variant of the Floyd-Steinberg error diffusion algorithm which consists of two 256K bytes LUTs and the calculation of the modified input requires 1 addition and 2 table look-up operations per pixel processed.*

## Introduction

Error diffusion [1] is a popular technique for digital halftoning, especially when high fidelity and faithful reproduction of high frequency features such as edges are required. Error diffusion requires several multiplications and additions per pixel processed. This is considerably slower than halftoning methods of the point operation type, such as the blue noise mask [2], which require one comparison per pixel processed and memory storage for the dither mask.

In Ref. [3] a fast error diffusion algorithm is proposed by replacing multiplication operations and quantization operations with look-up tables (LUT). In this letter, we propose another fast LUT-based error diffusion algorithm which offers more flexibility in the size of the LUTs. This allows for a more optimal tradeoff between speed, halftone image quality, hardware complexity and parallelizability. Furthermore, the proposed algorithm allows the precision in which each error term is calculated to be varied, allowing yet more flexibility in the design. The ultimate goal is to create neighborhood operation halftoning algorithms with speeds approaching those of point operation halftoning algorithms. We illustrate this by showing an implementation of the Floyd-Steinberg error diffusion algorithm where the computation of the modified input uses 2 LUTs and 1 addition per pixel processed.

## Error diffusion halftoning

The error diffusion algorithm can be described by the following steps. For each pixel, the current pixel value is quantized to produce the output halftone pixel value. The error at the current pixel, which is the difference between the pixel value and the output value is then distributed to neighboring pixels. More precisely, denoting the pixel value, the output value and the error value of a pixel $p$ as $v(p)$, $o(p)$ and $e(p)$ respectively, we compute at each pixel $p$:

1. $o(p) = Q(v(p))$
2. $e(p) = v(p) - o(p)$
3. $v(q) \leftarrow v(q) + w(p,q)e(p)$ for each pixel $q$ in the neighborhood of $p$

where $Q$ is the quantization function. The weights $w(\cdot,\cdot)$ are generally nonnegative[1] and sum to one, i.e $\sum_q w(p,q) = 1$ for each $p$. The weights are also referred to as the error diffusion kernel. Furthermore, in general the weights are shift-invariant, i.e. $w(p,q)$ depends only on the difference between $p$ and $q$, and we can replace the notation $w(p,q)e(p)$ with $w(i)e(p)$ where $q - p = i \in N$ and $N$ is a neighborhood of the origin. Many sets of weights have been proposed [5, 6], each with different characteristics such as simple hardware implementation, less anisotropic artifacts, etc.

### 1D LUT-based error diffusion algorithm

In Ref. [3] a fast implementation of error diffusion is proposed where the multiplications with the weights (step 3 above) are replaced by a one-dimensional LUT. In particular, at each pixel, the error $e(p)$ is multiplied with several weights $w(q-p)$ and thus a LUT is constructed which is indexed by $e(p)$ and produces the set of values $w(i)e(p)$ for $i \in N$ as output.

Let us assume that each error value $e(p)$ is represented as a $k$-bit number[2] and its weighted value $w(i)e(p)$ is represented as an $m$-bit number. Furthermore, the number of elements in the kernel is denoted by $n$, i.e. $|N| = n$. Then the LUT is indexed by $k$ bits and the size of the LUT is $2^k nm$ bits.

For instance, consider the Jarvis error diffusion algorithm [7] where $n = 13$ and the kernel is given by

$\frac{1}{48}$

| | | | | |
|---|---|---|---|---|
| | | × | 7 | 5 |
| 3 | 5 | 7 | 5 | 3 |
| 1 | 3 | 5 | 3 | 1 |

Here '×' denotes the current pixel position. As an example, if $e(p)$ and $w(i)e(p)$ are represented by 8 bits, ($k = m = 8$), then the LUT for the Jarvis algorithm is of size $256 \times 13 \times 8$ bits or 3328 bytes. The number of operations needed per processed pixel $p$ to propagate the error i.e. implement $v(q) \leftarrow v(q) + w(q-p)e(p)$ is 1 table look-up and $n$ additions.

**Remark** The method of replacing the multiplication of a single number with multiple fixed weights with a single LUT is also used to speed up convolution [8].

[1] See [4] for an algorithm where some of the weights are negative.

[2] Since the errors $e(p)$ can be negative and larger in magnitude than the pixel values $v(p)$, we require that the values of $e(p)$ and $w(i)e(p)$ are properly scaled and/or translated to fit into a $k$-bit (and $m$-bit resp.) fixed point number representation which can be signed or unsigned.

We see that the size of the LUT is mainly dictated by the bitdepths of the various numbers and the number of weights in the kernel. The first 2 steps, i.e. the calculation of $o(p)$ and $e(p)$, can also be replaced by two 1-D LUTs, but we will not focus on this part here as the main speed improvement is due to speeding up the multiplication operations. In the next section we propose a LUT-based error diffusion algorithm for which there is more flexibility in choosing the size of the LUT.

## A novel LUT-based error diffusion

An alternative but equivalent way to describe error diffusion is the following. At each pixel $p$, the modified input $M(p)$ is computed by adding the pixel value to weighted errors from pixels in the neighborhood of the current pixel. The output value is computed by quantizing the modified input value and the error is computed as the difference between the output value and the modified input. More precisely, the algorithm performs the following steps:

1. $M(p) = v(p) + \sum_{i \in N} w(i)e(i+p)$
2. $o(p) = Q(M(p))$
3. $e(p) = M(p) - o(p)$

For example, the Shiau-Fan error diffusion algorithm [9] has $n = 5$ elements in the kernel:

$\frac{1}{16}$

| | | | | |
|---|---|---|---|---|
| | 4 | 2 | 1 | 1 |
| 8 | × | | | |

In contrast to the interpretation in the previous section where errors are propagated to *future* pixels, here the modified error is computed by adding errors from *previous* pixels. Therefore the kernel in this interpretation is shown rotated 180° from the way error diffusion kernels are usually given in the literature. In the proposed LUT-based error diffusion algorithm, we replace the computation of $\sum_{i \in N} w(i)e(i+p)$ with one or more LUTs. Assume that $e(i+p)$ is represented using $k$ bits. Then the input of the LUT are the $n$ error values $e(i+p)$ and consist of $nk$ bits. In particular, consider the following diagram of the $nk$ bits corresponding to the $n$ error values (Fig. 1). The output of the LUT is the $m$-bit number $\sum_{i \in N} w(i)e(i+p)$. The value of $M(p)$ is equal to the sum of the output of the LUT and the current input pixel.[3] If this LUT of size $2^{nk}m$ bits can fit into memory, then we obtain an error diffusion algorithm where the calculation of $M(p)$ takes 1 table look-up and 1 addition. However, in current computer architectures, this LUT is too large to fit in memory, especially fast memory such as cache memory closest to the CPU. To illustrate, for Jarvis error diffusion with 8-bit precision for the errors, $k = m = 8$, and $n = 13$ would result in a LUT of $2^{104}$ bytes or 16 tebi-exbibytes. Therefore we split up this LUT into several LUTs, and sum up the outputs from these LUTs to obtain $\sum_{i \in N} w(i)e(i+p)$. This process is valid because of the linearity of the computation.

To create the various LUTs, the bits in Figure 1 are partitioned into several blocks as shown in Fig. 2. Even though the blocks in Fig. 2 are shown as connected shapes, this is not necessary. Each of these blocks corresponds to a linear combination of subsets of bits of some error values $e(i+p)$. Thus each LUT takes the bits in each block as input and output the corresponding linear combination. By linearity, we can add the outputs of these LUTs and obtain the full linear combination. Since we are adding outputs from LUTs, each of which can cover a different range, the resulting $\sum_{i \in N} w(i)e(i+p)$ can have higher precision than each individual output. Even though Figure 2a illustrates a general partition with arbitrary shapes, certain partitions result in simpler implementations. For instance, consider the partition of the $nk$ bits shown in Fig. 2b. Each LUT computes the linear combination of a bitplane (or several adjacent bitplanes) of all the error values. It is easy to see that the outputs of the LUTs differ only by a shift in the bits, i.e. a factor of $2^v$. Thus we can replace the various LUTs by a single LUT and shift the bits of the output appropriately depending on which bitplanes are used as input. As in [3], use of general weights $w(i)$ does not increase hardware complexity versus special weights such as $\frac{1}{2}$ or $\frac{1}{4}$.

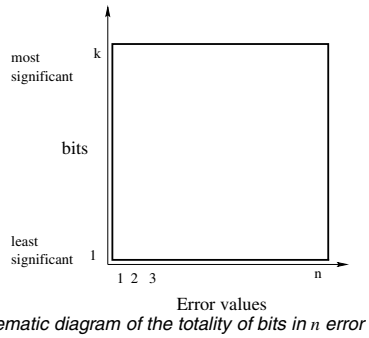



***Figure 1.*** *Schematic diagram of the totality of bits in $n$ error values.*

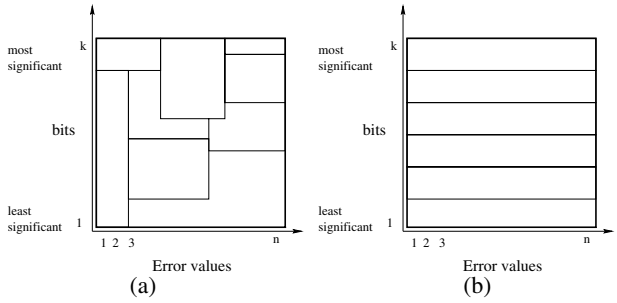



***Figure 2.*** *Schematic diagram of partitions of $nk$-bits of $n$ error values. (a) general partition. (b) bitplane partition.*

As an example, consider the following implementation of LUT-based Jarvis error diffusion. In this case $n = 13$ and let us assume that $k = m = 8$. We partition as in Fig. 2b into 8 LUTs which is implemented in a single LUT of size $2^{13} \times 8$ bits = 8192 bytes. Computation of $M(p)$ takes 8 look-up operations, 7 shift operations and 8 additions operation. If we use 8 separate LUTs occupying a total of 64K bytes, then the 8 table look-up operations can be done in parallel and the time to compute $M(p)$ is 1 table look-up operation and 8 additions. This is compared with the algorithm in [3] which requires a LUT of 3328 bytes, 1 table look-up operation and 13 additions per pixel processed. One feature of the proposed implementation is the flexibility in the choice of size of the LUT which depends on how the $nk$ bits are partitioned. For instance, by partitioning Fig. 1 into 4 bit planes, each containing two adjacent bits we get 4 LUT tables with $2^{2n}$ entries each. Again these 4 LUT tables differ from each other by

[3] Similar to before, the input and the output of the LUT are expressed in appropriate number formats.

a shift of the bits and they can be replaced by a single LUT. As in [3], the calculation of $o(p)$ and $e(p)$ can be replaced with 1-D LUTs.

Another feature of this implementation is that the calculation bit depth of each error value can be variable. The calculation bit depth is defined as the number of bits used in the computation. This is different (and possibly smaller) than the actual number of bits used to the store the error value in memory. Furthermore, whereas the actual number of bits used for a single error value is the same, its calculation bit depth can change depending on the corresponding weight that it is multiplied with. For example, for small weights the calculation bit depth can be smaller than the calculation bitdepth for large weights. On the vertical axis of Figs. 1 and 2, $k$ is in fact the calculation bitdepth of the error values. In the case of variable calculation bitdepth, the set of bits of the errors is shown schematically in Fig. 3.

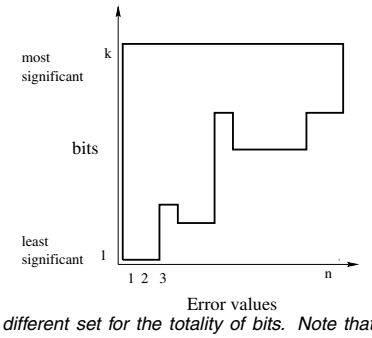



***Figure 3.*** *A different set for the totality of bits. Note that the calculation bitdepth of the different error values are different.*

Consider the following implementation of the Shiau-Fan error diffusion algorithm. Assume that the calculation bitdepth of an error value is given in parentheses next to the corresponding weight in the following diagram:

| | $\frac{4}{16}$ (8) | $\frac{2}{16}$ (6) | $\frac{1}{16}$ (4) | $\frac{1}{16}$ (4) |
|---|---|---|---|---|
| $\frac{8}{16}$ (8) | $\times$ | | | |

It seems reasonable that the calculation bitdepth should increase with the magnitude of the weight. We construct two 32K bytes LUTs with each LUT computing using half the bitdepth assigned to each error value. Then the calculation of $M(p)$ requires 2 table look-up and 2 addition operations. The output from this implementation (Fig. 5) is qualitatively the same as the original Shiau-Fan algorithm (Fig. 4).

In another example, we use the following calculation bitdepth distribution for the weights:

| | $\frac{4}{16}$ (4) | $\frac{2}{16}$ (3) | $\frac{1}{16}$ (2) | $\frac{1}{16}$ (2) |
|---|---|---|---|---|
| $\frac{8}{16}$ (5) | $\times$ | | | |

We create a single 64K bytes LUT and calculate $M(p)$ using 1 table look-up operation and 1 addition. The output of this halftoning algorithm is shown in Fig. 6. The halftone image suffers some quality loss and has increased contrast.

So far, the LUT table is used to compute $\sum_{i\in N} w(i)e(i+p)$ and this is added to the input pixel $v(p)$ to obtain the modified input $M(p)$. Another variant of this algorithm is to have $v(p)$ as

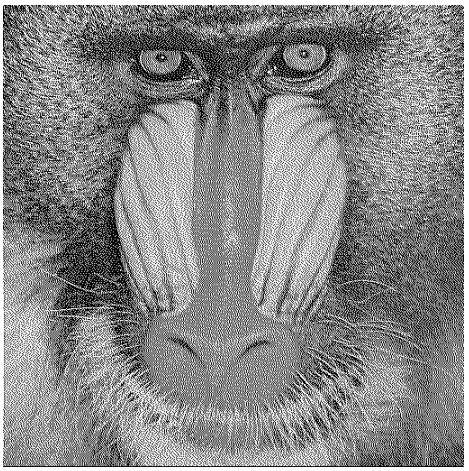


***Figure 4.*** *Shiau-Fan error diffusion.*

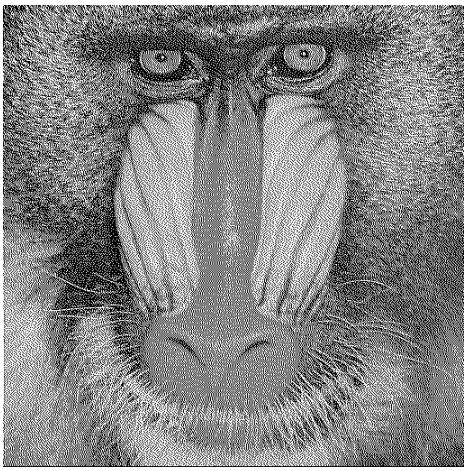


***Figure 5.*** *Shiau-Fan error diffusion using two LUTs.*

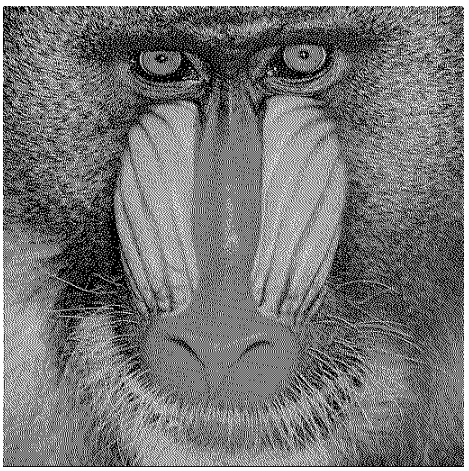
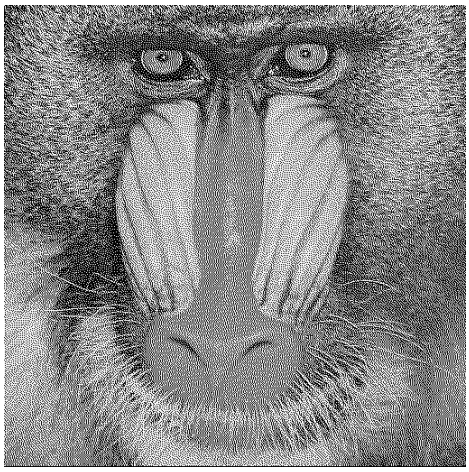


***Figure 6.*** *Shiau-Fan error diffusion using a single LUT.*

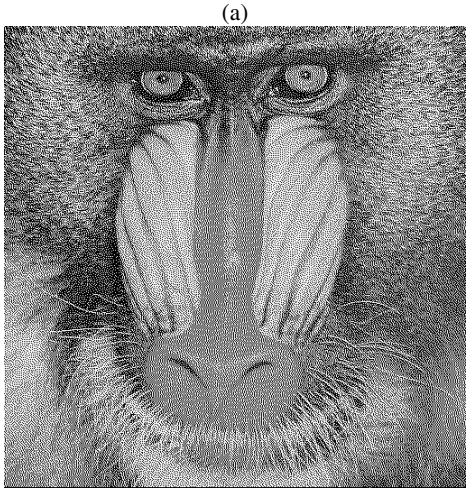
part of the input to the LUT and have the LUT compute $M(p)$ directly. This adds another level of flexibility to the tradeoff equation. For example, consider Floyd-Steinberg error diffusion [1] with weights and calculation bitdepth distribution:

| | | |
|---|---|---|
| $\frac{1}{16}$ (6) | $\frac{5}{16}$ (8) | $\frac{3}{16}$ (6) |
| $\frac{7}{16}$ (8) | $\times$ (8) | |

The number in parentheses next to '$\times$' denotes the calculation bitdepth of the current pixel value $v(p)$. Using two LUTs to split up the bits, we obtain two 256K bytes LUTs whose outputs are summed up to obtain $M(p)$. If the LUTs are accessed in parallel, the time it takes to compute $M(p)$ is 1 table look-up operation and 1 addition operation. The halftone output is shown in Fig. 7b which is qualitatively the same as the original Floyd-Steinberg algorithm (Fig. 7a). The speed of this implementation approaches that of the blue noise mask dithering algorithm [2], which for large dither masks has similar memory requirements. This is significant since dithering algorithms are point operation algorithms and thus are inferior to error diffusion in terms of sharp details rendition.



(a)



(b)

***Figure 7.*** *Floyd-Steinberg error diffusion. (a) Original algorithm. (b) Implemented using two LUTs and only 1 addition operation is needed to compute the modified input $M(p)$.*

## Conclusions

We have presented a novel LUT-based error diffusion algorithm which allows for a flexible tradeoff between output image quality, hardware complexity and processing speed. In particular, quality can be traded off by varying the calculation bitdepth of the error values. Furthermore, the multiple LUTs used are amenable to parallel implementation. The above technique can be used in various variations of the error diffusion algorithms such as algorithms where the kernel depends on the input pixel value [4]. It can also be useful in algorithms where the computation of a linear combination of values needs to be sped up. This include halftoning algorithms such as Direct Binary Search [10] and other types of iterative halftoning algorithms [11].

One implementation issue is the repacking of the bits of the pixel and error values to form an index for the LUT. This is best incorporated into the hardware design as it requires little or no computation at all. An example of hardware implementation where bits are repacked to form an index to a LUT can be found in [12] where it is used to enable print quality enhancement of binary text data. With such hardware designs and fast enough memory, we can expect error diffusion to be performed at speeds comparable to point operation halftoning algorithms such as dither mask screening.

## References


[1] R. W. Floyd and L. Steinberg, "An adaptive algorithm for spatial grayscale," *Proceedings of the Society for Information Display*, vol. 17, no. 2, pp. 75–77, 1976.

[2] M. Yao and K. J. Parker, "Modified approach to the construction of the blue noise mask," *Journal of Electronic Imaging*, vol. 3, no. 1, pp. 92–97, 1994.

[3] H. R. Kang, "Fast error diffusion," in *Proceedings of SPIE*, vol. 4663, pp. 302–309, 2002.

[4] C. P. Tresser and C. W. Wu, "Target patterns controlled error management." US Patent 6006011, 1999.

[5] R. Ulichney, *Digital Halftoning*. Cambridge, MA: MIT Press, 1987.

[6] H. R. Kang, *Digital Color Halftoning*. SPIE Press, 1999.

[7] J. Jarvis, C. Judice, and W. Ninke, "A survey of techniques for the display of continuous tone pictures on bilevel displays," *Computer Graphics and image Processing*, vol. 5, pp. 13–40, 1976.

[8] G. Wolberg and H. Massalin, "Fast convolution with packed lookup tables," in *Graphics Gems IV*, pp. 447–464, Academic Press, 1994.

[9] J. Shiau and Z. Fan, "A set of easily implementable coefficients with reduced worm artifacts," in *Proceedings of SPIE*, vol. 2658, pp. 222–225, 1996.

[10] J. P. Allebach, "DBS: retrospective and future directions," in *Proceedings of SPIE*, vol. 4300, pp. 358–376, 2001.

[11] C. W. Wu, G. Thompson, and M. Stanich, "A unified framework for digital halftoning and dither mask construction: variations on a theme and implementation issues," in *Proceedings IS&T's NIP19: International Conference on Digital Printing Technologies*, pp. 793–796, 2003.

[12] M. J. Stanich, "Print-quality enhancement in electrophotographic printers," *IBM Journal of Research and Development*, vol. 41, no. 6, pp. 669–678, 1997.