 Copyright 2000, IS&T

Error Diffusion Using Linear Pixel Shuffling

Peter G. Anderson
Rochester Institute of Technology
Rochester, NY, USA

Abstract

Linear pixel shuffling error diffusion is a digital halftoning algorithm that combines the linear pixel shuffling (LPS) order of visiting pixels in an image with diffusion of quantization errors in all directions.

LPS uses a simple linear rule to produce a pixel ordering giving a smooth, uniform probing of the image. This paper elucidates that algorithm.

Like the Floyd-Steinberg algorithm, LPS error diffusion enhances edges and retains high-frequency image information. LPS error diffusion avoids some of the artifacts ("worms," "tears," and "checkerboarding") often associated with the Floyd-Steinberg algorithm. LPS error diffusion requires the entire image be available in memory; the Floyd-Steinberg algorithm requires storage proportional only to a single scan line.

1. Error Diffusion Halftone Production

The error diffusion algorithm transforms a gray scale image, `I`, with pixel values in the interval $[0.0, 1.0]$, to a black-and-white image, `B`, with values in $\{0, 1\}$. The following pseudocode describes error diffusion:

```
for every pixel position i,j in I
   if I[i][j] < 0.5
      then B[i][j] = 0
      else B[i][j] = 1
   error = I[i][j] - B[i][j]
   distribute the error among
   unprocessed neighbors of i,j
```

The order of pixel visitation generally takes the form of raster processing:

```
for ( i = 0; i < i_max; i++ ) {
   for ( j = 0; j < j_max; j++ ) {
      process pixel i,j
   }
}
```

The input and output images dimensions are `i_max` by `j_max`. Floyd and Steinberg's error diffusion algorithm [4] follows this pixel ordering and distributes the error to four unprocessed neighbors of `I[i][j]` according to the kernel D_{FS}:

$$D_{FS} = \frac{1}{16}\begin{bmatrix} & P & 7 \\ 3 & 5 & 1 \end{bmatrix} \quad (1)$$

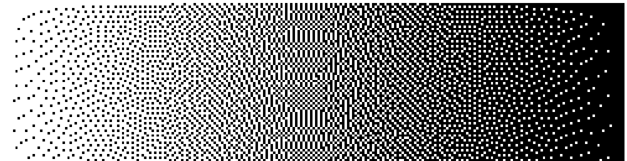

Figure 1: Floyd-Steinberg error diffusion. Notice the ìwormî artifacts at the ends, and ìtearingî in the middle. (All the ramp images are 64×256 pixels.)

where P denotes the pixel currently being processed.

```
I[i][j+1]     += error * 7/16
I[i+1][j-1]   += error * 3/16
I[i+1][j]     += error * 5/16
I[i+1][j+1]   += error * 1/16
```

This process produces bilevel images with visual appearance capturing the full range and detail of the original image. This is particularly effective in case the original image has a lot of detail. The resulting images do, however, often contain "worm" artifacts in very dark and very light regions, and a "tearing" or "checkerboarding" where the image's original gray value was slowly varying around 0.25, 0.5, or 0.75; see Fig. 1 (our ramp figures are presented here using enlarged pixels–approximately 80dpi—in order to illustrate the results of the algorithms and avoid transformations caused by unknown printing processes; the proper viewing distance is 4–5 feet). Floyd-Steinberg error diffusion was also used to render the "Woman with French Horn" in Fig. 2 (approx. 130dpi).

2. Linear Pixel Shuffling

Linear pixel shuffling (LPS) is a method for visiting image pixels distributed evenly all over an image. LPS uses a simple linear rule, given by matrix multiplication (modulo a parameter depending on the image size)

$$\begin{bmatrix} i \\ j \end{bmatrix} = M \begin{bmatrix} x \\ y \end{bmatrix} \quad (2)$$

The original image processing pseudocode uses Eq. (2), becoming:

Figure 2: *Floyd-Steinberg error diffusion of "Woman with French Horn" (image courtesy of Heidelberger Druckmaschinen A. G.).*

```
for ( x = 0; x < N; x++ ) {
   for ( y = 0; y < N; y++ ) {
      multiply M * (x,y)'
      to obtain (i,j)';
      process pixel i,j
   }
}
```

The matrix M and the parameter N are described below.

Define the Fibonacci-like sequence G by the recurrence:

$$\begin{aligned} G_0 &= 0 \\ G_1 &= 1 \\ G_2 &= 1 \\ G_{n+1} &= G_n + G_{n-2} \text{ for } n \geq 2 \end{aligned} \quad (3)$$

Terms G_0 through G_{14} of this sequence are

$$0, 1, 1, 1, 2, 3, 4, 6, 9, 13, 19, 28, 41, 60, 88$$

We also need this sequence with negative subscripts (the definition allows us to work backwards in the subscripts easily). Terms G_{-1} through G_{-14} are

$$0, 1, 0, -1, 1, 1, -2, 0, 3, -2, -3, 5, 1, -8$$

For our algorithm, we round the image size up to a square of side G_n (this is the N in the above pseudocode) and ignore pixels that fall outside the actual image.

Define a $G_n \times G_n$ table T with entries defined as

$$T_{pq} = (pG_{n-2} + qG_{n-1}) \pmod{G_n} \quad (4)$$

Because

$$\gcd(G_{n-2}, G_{n-1}, G_n) = 1 \quad (5)$$

for all n, every number in $0, 1, 2, \cdots, G_n - 1$ occurs in T exactly G_n times. Here is a portion of the 88×88 table for $0 \leq p, q < 13$.

0	60	32	4	64	36	8	68	40	12	72	44	16
41	13	73	45	17	77	49	21	81	53	25	85	57
82	54	26	86	58	30	2	62	34	6	66	38	10
35	7	67	39	11	71	43	15	75	47	19	79	51
76	48	20	80	52	24	84	56	28	0	60	32	4
29	1	61	33	5	65	37	9	69	41	13	73	45
70	42	14	74	46	18	78	50	22	82	54	26	86
23	83	55	27	87	59	31	3	63	35	7	67	39
64	36	8	68	40	12	72	44	16	76	48	20	80
17	77	49	21	81	53	25	85	57	29	1	61	33
58	30	2	62	34	6	66	38	10	70	42	14	74
11	71	43	15	75	47	19	79	51	23	83	55	27
52	24	84	56	28	0	60	32	4	64	36	8	68

A particularly nice feature of the table T is that values which are numerically close are physically distant. Examine a 7×7 block centered at one of the zeros in T:

49	21	81	53	25	85	57
2	62	34	6	66	38	10
43	15	75	47	19	79	51
84	56	28	0	60	32	4
37	9	69	41	13	73	45
78	50	22	82	54	26	86
31	3	63	35	7	67	39

Notice that the smaller numbers are not close to 0. Because T was constructed using a simple linear rule, this phenomenon holds for any value, not just zero; it is even more visible for larger values of G_n.

The LPS pixel selection algorithm works by first visiting the elements in the $G_n \times G_n$ square that correspond to the positions (i, j) for which $T_{ij} = 0$, followed by positions (i, j) for which $T_{ij} = 1$, and so on. We may express this in the processing pseudocode:

```
for ( x = 0; x < N; x++ ) {
      process all N pixels i,j
      such that T[i][j] = x
}
```

The matrix M that achieves this is

$$M = \begin{bmatrix} G_{-n+1} & G_{n-3} \\ G_{-n} & G_{n-2} \end{bmatrix} \quad (6)$$

For example, using $\mathtt{N} = G_{14} = 88$

$$M = \begin{bmatrix} G_{-13} & G_{11} \\ G_{-14} & G_{12} \end{bmatrix} = \begin{bmatrix} 1 & 28 \\ -8 & 41 \end{bmatrix} \quad (7)$$

Copyright 2000, IS&T

2.1. Mathematical Justification of the LPS Algorithm

The above algorithm processes all $G_n \times G_n$ image pixels for the following reason. Let

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} G_{-n+1} & G_{n-3} \\ G_{-n} & G_{n-2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{8}$$

We will show that

$$T_{ij} = x \tag{9}$$

Multiplying out the above matrix formula, we have

$$\begin{aligned} T_{ij} &= iG_{n-2} + jG_{n-1} && (10) \\ &= (G_{-n+1}x + G_{n-3}y)G_{n-2} \\ &\quad +(G_{-n}x + G_{n-2}y)G_{n-1} && (11) \\ &= (G_{-n+1}G_{n-2} + G_{-n}G_{n-1})x \\ &\quad +(G_{n-3}G_{n-2} + G_{n-2}G_{n-1})y && (12) \\ &= (G_{-n+1}G_{n-2} + G_{-n}G_{n-1})x && (13) \\ &\quad +(G_{n-3} + G_{n-1})G_{n-2}y && (14) \end{aligned}$$

The values in Eq. (8–14) are all modulo G_n. One of the factors in (14) satisfies

$$G_{n-3} + G_{n-1} = G_n \equiv 0 \pmod{G_n} \tag{15}$$

by (3); so T_{ij} is independent of y. The coefficient of x in (13) satisfies

$$G_{n-2}G_{-n+1} + G_{n-1}G_{-n} \equiv 1 \pmod{G_n} \tag{16}$$

for all n, because

$$G_{n-2}G_{-n+1} + G_{n-1}G_{-n} + G_nG_{-n+2} = 1 \tag{17}$$

for all n; this is easily established by mathematical induction. Finally, because three consecutive terms of the G sequence are relatively prime (5), $M \begin{bmatrix} 0 \\ y \end{bmatrix}$ takes G_n distinct values for $0 \le y < G_n$. Linearity allows us to conclude that $\begin{bmatrix} i \\ j \end{bmatrix}$ ranges over all $G_n \times G_n$ pixel positions.

2.2. Other Applications of LPS

There are a wide variety of applications for LPS; most of them are explained in detail in [2]. Here are several that pertain directly to image processing.

LPS was originally developed to aid development of computer graphics image synthesis, especially fractal creation (see [1]). By rendering pixels in the LPS order, a low resolution indication of the eventual picture is almost immediately visible. Slow computers or ambitious programmers otherwise produce images that can take hours to see—starting with the extremely boring topmost rows of pixels.

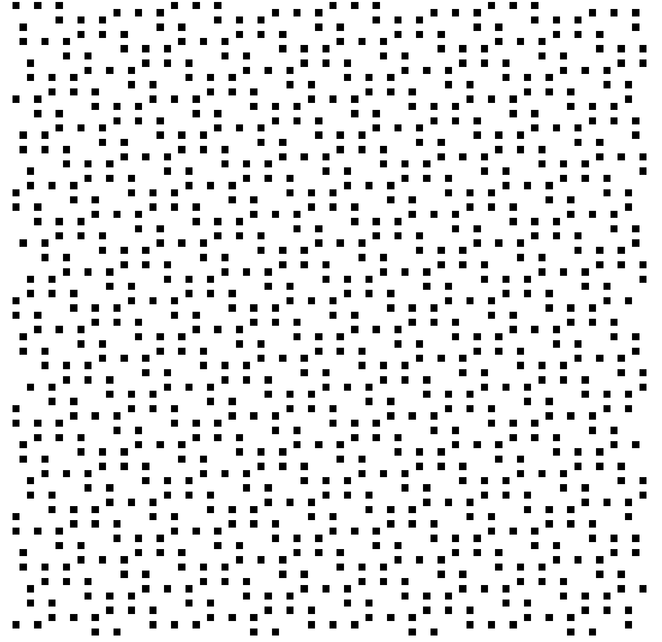

Figure 3: *An* 88×88 *image with the rst 968 (12.5%) of the pixels marked black.*

The pixel ordering for image rendering suggests an image file format. The initial portion of such a file gives a low resolution image; subsequent portions fill in the details.

Because LPS is *linear*, the pixels that are rendered earliest can be magnified ("fat pixels") to cover the entire image area very early. Subsequent pixels can be magnified by smaller factors, eventually not magnified at all. If there are constant patches in the image, the later pixels often make no difference to the image appearance—they do not need to be rendered or stored at all. This observation leads to a lossless image compression technique for text and line-drawing images that is competitive with run-length encoding but has the useful property that a prefix of the compressed file is a lossy compression of the entire image.

The results of image morphology operations (dilation, erosion, opening, closing) can be rapidly approximated by performing the basic operations in LPS order. For some applications, an approximation that takes, say, 15% of the total time may be sufficient.

The table T can be used as a halftone mask. Fig. 3 shows an 88×88 image with the first 968 of the pixels marked. This is the dot pattern associated with an LPS mask at gray level 0.125 (where 1.0 indicates black).

Images can be searched to locate objects ("Where's Waldo?") or gather image statistics, such as histogram estimation. An infinite version of LPS is appropriate for Monte Carlo integration.

Copyright 2000, IS&T

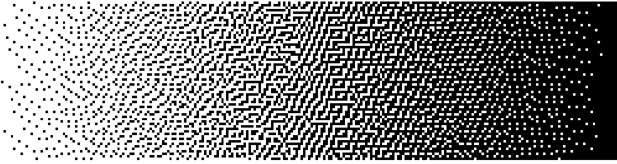

Figure 4: *The ramp rendered using LPS error diffusion.*

3. LPS Error Diffusion

By visiting the pixels uniformly all over the image, LPS allows us to diffuse the errors in all directions. For example, we can use the kernel which John Szybist investigated for his Computer Science Master's Project [3]:

$$D_{LPS} = \frac{1}{32}\begin{bmatrix} & 1 & 1 & 1 & \\ 1 & 2 & 3 & 2 & 1 \\ 1 & 3 & P & 3 & 1 \\ 1 & 2 & 3 & 2 & 1 \\ & 1 & 1 & 1 & \end{bmatrix} \qquad (18)$$

and distribute the error of the processed pixel P to 20 of its neighbors.

In order not to loose any of the dispersed error, we need to keep track of any neighborhood of a zero value in T (a 5×5 neighborhood in this case) and note the positions in the kernel corresponding to already quantized pixels. We thus update the diffusion kernel after each completed inner loop (the y-loop in our pseudocode), potentially replacing some non-zero coefficient with a zero and updating the divisor (initially 32 here) correspondingly.

In addition to the Szybist filter (18) used for Figs. 4 and 5 we can use a wide variety of kernels; several are shown in Fig. 6. Ramps rendered using these kernels are shown in Fig. 7.

4. Comparisons and Conclusions

LPS error diffusion does eliminate worms, tearing, and checkerboarding artifacts. Unfortunately, LPS halftoned images appear to be mottled in comparison to Floyd-Steinberg halftoned images.

The LPS halftoning algorithm requires that the entire image be present in memory, in order to diffuse error in all directions. Floyd-Steinberg only requires one or two scan lines (depending on coding cleverness, to diffuse errors only to the right and down). The number of operations in both cases is linear in kernel size and number of image pixels.

The texture variations of images rendered by various LPS halftoning kernels may present opportunities for achieving good dot patterns of ink on paper that eludes most current digital halftone systems.

Figure 5: *Linear pixel shuf ing error diffusion.*

References

[1] Peter G. Anderson. Fast rendering. *Computer Language*, Feb 1993.

[2] Peter G. Anderson. Advances in linear pixel shuffling. In G. E. Bergum, A. N. Philippou, and A. F. Horodam, editors, *Conference on Fibonacci Numbers and Their Applications*, pages 1–21, Pullman, Washington, July, 1994.

[3] John Szybist. An error diffusion algorithm based on linear pixel shuffling. Computer science master's project, Rochester Institute of Technology, 1997.

[4] Robert Ulichney. *Digital Halftoning*. The MIT Press, 1987.

Copyright 2000, IS&T

flat-3:

```
1 1 1
1 P 1
1 1 1
```

flat-5:

```
1 1 1 1 1
1 1 1 1 1
1 1 P 1 1
1 1 1 1 1
1 1 1 1 1
```

flat-7:

```
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 P 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
```

ring-5:

```
1 1 1 1 1
1 0 0 0 1
1 0 P 0 1
1 0 0 0 1
1 1 1 1 1
```

ring-7:

```
1 1 1 1 1 1 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 P 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 1 1 1 1 1 1
```

cross:

```
0 0 1 0 0
0 0 1 0 0
1 1 P 1 1
0 0 1 0 0
0 0 1 0 0
```

Figure 6: *A variety of kernels for LPS error diffusion.*

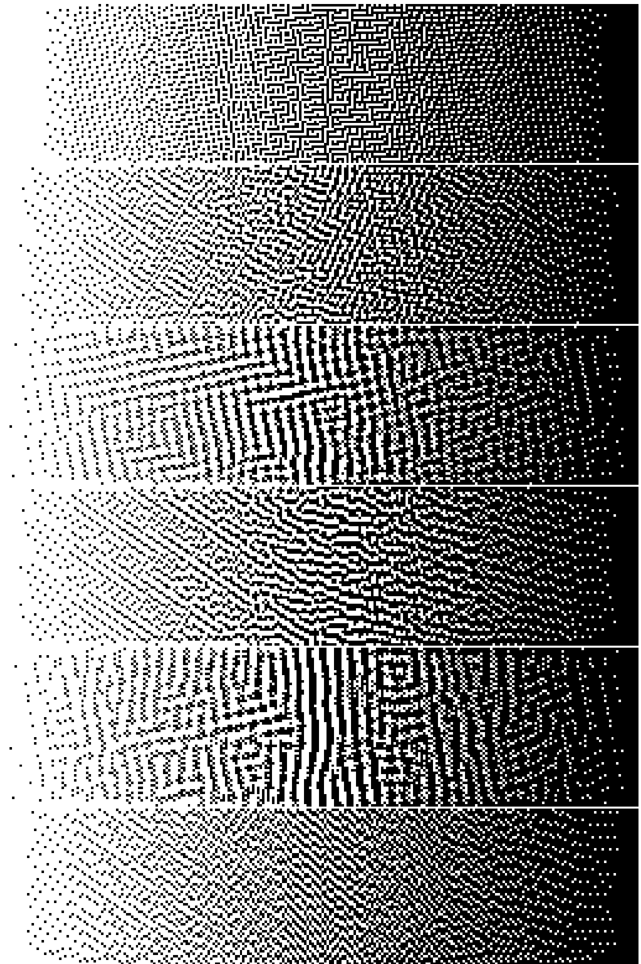

Figure 7: *LPS error diffusion using kernels: at-3, at-5, at-7, ring-5, ring-7, and cross.*