# Image Dithering

**Dr.N.Sairam & Dr.R.Seethalakshmi**
**School of Computing,**
**SASTRA Univeristy,**
**Thanjavur-613401**.

## Contents

## 1. Image Dithering

Dithering or half-toning is a technique that is used to represent a gray scale image on a printer, computer monitor or other bi-level displays. This technique is called as a error diffusion technique. In this context, a parallel error diffusion technique is getting introduced. Dithering is also known as a compression and low bandwidth compression technique. There are several dithering techniques. Among this the simple and quality based one is the error diffusion algorithm. The algorithm proposed by Floyd and Steinberg is discussed here.

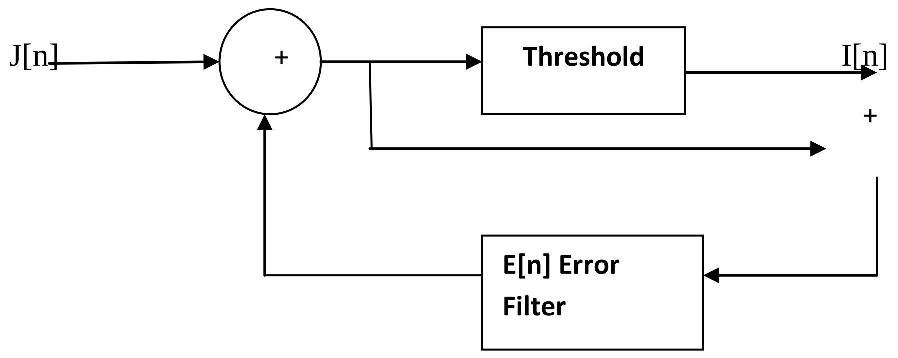

Figure 1 Floyd-Steinberg dithering process

Pixels J[n] of continuous tone digital image are processes in a linear fashion, left-to-right and top-to-bottom. The algorithm compares the gray level intensity value of the current pixel with the threshold (128). If the input gray scale value is greater than the threshold, the pixel is considered black and its value I[n] is 1, else the pixel is white and I[n] is 0. The error that occurs is nothing but the difference between the original gray scale value and the threshold. To reduce the error the error is distributed to the four unprocessed pixels as shown in Figure 2 below.

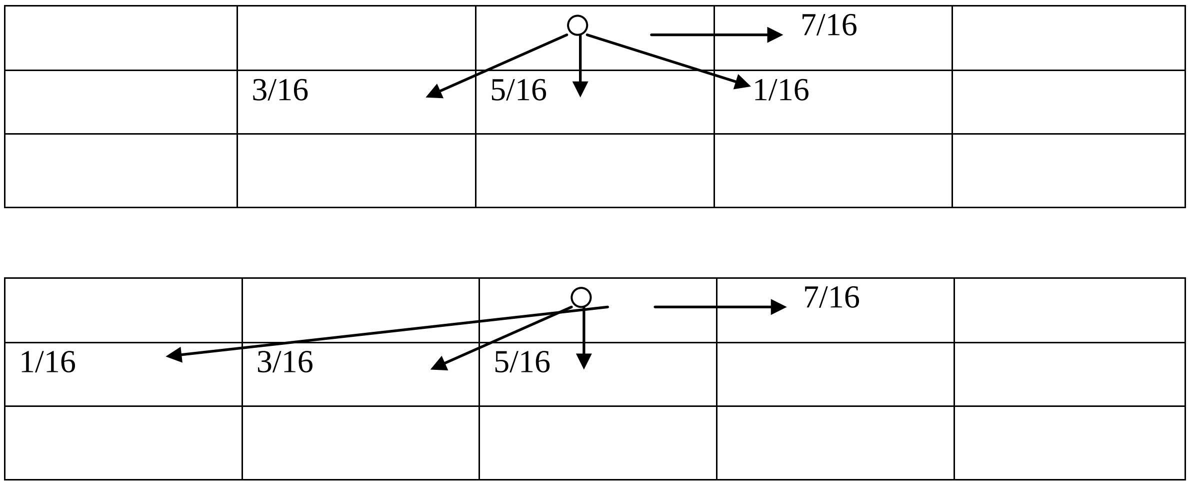

Figure 2: (a) Distributing error to four neighboring pixels and(b) its modified form

The new matrix after distribution is used to improve the quality of the dithered image without increasing the overhead. Dithering is a time consuming process. The straightforward implementation of dithering requires four floating point operations and five memory accesses. Therefore it is computationally quite expensive. Parallelizing the dithering process may lead to faster printer and larger monitors. The scientists thought that Floyd and Steinberg method has some overlaps and can be parallelized. An example will explain this in detail. Consider the prefix sum application. An array can be decomposed into sub arrays and each processor can work on the sub arrays to calculate the partial prefix sum and total sum in parallel.

In the case of dithering a particular pixel can be processed only after all the dependent pixels can be processed. In the parallel implementation, a linear array is embedded in a two dimensional array of processing elements (PE). In a linear array each PE has the input and output and can be communicating with the left and right neighbor. In the two dimensional array, there are four communicating neighbors. They are left, right, top, bottom neighbors, except the corner PEs. A parallel machine is considered for implementing the parallel dithering. Each PE will be communicating with the 8 connected neighbors as well. Pixels are scheduled for dithering at process times with a pattern as shown in figure 3.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |

Figure 3 Image Slices and processing time for each pixel

The algorithm works as follows. Let $k =$ n-1/3 where n is the number of rows of image. The three uppermost left pixels are processed and their dithering errors are calculated by the kth processor in the linear array called the starting processor. After these three steps the kth processor sends the appropriate fractions of the errors to its two neighbors $P_{k+1}$ and $P_{k-1}$. It then continues with the processing of the 2nd, 3rd and 4th pixel of the second row of the image. In the meantime $P_{k+1}$ and $P_{k-1}$can be processed with the image slicesof their own. When reading pixels directly from memory, the main implementation difficulty is to have each processor $P_i$ knows which pixel to process at every step and where to send the error fractions.It turns out that processor $P_{k+\alpha}$to the right of the starting processor $P_k$at time $\tau>=2\alpha+1$ and processes pixels $(\tau, 3\alpha+\tau)$, $(\tau,3\alpha+\tau+1)$ and $(\tau,3\alpha+\tau+2)$. Like this it is also done in the left of the processor. The processor to the left of $P_k$at time $\tau>=2\alpha-1$ processes pixels $(2\alpha+\tau,-\alpha+\tau)$,$(2\alpha+\tau,-\alpha+\tau+1)$ and $(2\alpha+\tau,-\alpha+\tau+2)$.

In parallelizing, it is required that the width of the area should be atleast 3. For widths 1 or 2 pixels each processor communicates with both of its neighbors. Since most of the time communication is costlier in parallel systems than computation, it is required the processor communicates with one of the neighbors at each step. Based on the communication to computation ratio the width of the slanted area is finalized. For width >= (n+m)/3 each processor evaluates to a total of 3n pixels where n is the number of rows in the image. In the more realistic case p<(n=m)/3 the image is decomposed into p wider slanted areas each of width w=(n+m)p.

Given a large p the running time of the algorithm is T(n,m) = 2n+m-2=2(n-1)+m which is smaller the corresponding sequential implementation of 9.n.m. For smaller p the running time of the algorithm is T(n,m)= (w-1)(n-1)+m.