The void-and-cluster method for dither array generation

Robert Ulichney

Digital Equipment Corporation
Maynard, MA 01754-2571

ABSTRACT

Halftoning to two or more levels by means of ordered dither has always been attractive because of its speed and simplicity. However, the so-called recursive tessellation arrays in wide used suffer from strong periodic structure that imparts an unnatural appearance to resulting images.

A new method for generating homogeneous ordered dither arrays is presented. A dither array is built by looking for voids and clusters in the intermediate patterns and relaxing them to optimize isotropy. While the method can be used for strikingly high quality artifact-free dithering with relatively small arrays, it is quite general; with different initial conditions the familiar recursive tessellation arrays can be built.

This paper presents the algorithm for generating such arrays. Example images are compared with other ordered dither and error diffusion-based techniques.

1. Ordered Dither

Halftoning by means of the point operation of ordered dither is an old and established idea. Among the many methods for dithering an image [1,2,3], ordered dither is the easiest to implement as it does not require processing or storage of neighboring pixels. This makes for fast software or very simple hardware designs. These features are especially important for motion video rendering.

For rendering a continuous-tone image to 2 levels, as in the case of printing on a bitonal device, pixels from the input image are compared with elements from an dither array or matrix. The output is 1 if the input is greater than the dither threshold, and 0 otherwise. This can be extended to multilevel output dithering as well, as outlined in section 6. The nature and quality of the resulting image depends only on the precomputed dither array, and not the the ordered dither algorithm.

The "printer's" screen, used in rendering photographs for newspapers for example, is a form of ordered dither where the thresholds in the dither array are arranged in spiral patterns. Such "clustered-dots" are needed in offset printing applications where the physical area of an individual pixel is too small to hold ink on the printing plate. Where such single-pixel constraints are not an issue, as in many electronic displays, "dispersed-dot" dithering is preferred as the effects of the halftoning are less noticeable.

Perhaps the most widely used patterns for dispersed-dot ordered dither are those popularized by a paper by Bayer in 1973 [4]. These patterns were found to be a subset of those produced by the "method of recursive tessellation" [5]. While these patterns are indeed homogeneous, they suffer from rigid regular structures that introduce an artificial processed look to the resulting images.

0-8194-1146-9/93/$6.00

The use of randomness to break up the rigid regularity of clustered-dot ordered dither was perhaps first introduced by Allebach in 1976 [6,7]. Mitsa and Parker [8] present a way to produce an ordered dither array that will generate patterns with given spatial frequency domain characteristics. This was an attempt to mimic the "blue noise" patterns achieved with more compute intensive error diffusion algorithms described in [3, ch. 8]. Related to this is an algorithm for creating a binary pattern with any given power spectrum [9].

This paper presents a new method for generating dispersed-dot ordered dither arrays as small as 32 by 32 that result in very high quality patterns, free of directional artifacts. Like the method of recursive tessellation, dither array creation occurs entirely in the spatial domain.

2. Overview of New Method

We will create a Dither Array of size M by N, where M and N are any positive integer. Note that these sizes are not restricted to be equal or powers of 2. While any ordered Dither Array can be used to dither to a multilevel output, it is heuristicly most convenient to consider the problem of dithering to only 2 output levels, 0 or 1, and extend to the multilevel case later.

The Dither Array is a matrix with each element assigned a value corresponding to the order in which that location in an input image is to be assigned a 1. The elements are thus ordered, and assigned a unique integer from 0 to $(MN-1)$, which will be referred to as a *Rank*; As with other ordered dither arrays, these values are normalized at render time to match the range of input and output values. The M by N array is periodic in two-space. That is, it is modulo M in one dimension, and modulo N in the other.

To facilitate the ranking of elements in the Dither Array, a second array or matrix of exactly the same dimensions as the Dither Array is used, which we will call the Prototype Binary Pattern. The Prototype Binary Pattern, a matrix of 1's and 0's, is the output image that would correspond to a constant input image of value equal to the number of 1's in the Prototype Binary Pattern divided by the size, MN. (It is assumed here that the input value can be any real number between 0 and 1.0.) An input value of 0 would then correspond to an output of all 0's, and an input value of 1.0 would correspond to an output of all 1's. The important cases are those in between; the goal is to produce patterns of 1's and 0's that are homogeneously distributed, and free from disturbing structures.

The creation of our Dither Array will thus require the simultaneous processing of 2 arrays: the Prototype Binary Pattern and the Dither Array itself.

The order in which 1's are added to or removed from the Prototype Binary Pattern is reflected in the value, or *Rank*, of elements in the Dither Array. The *Rank* of a Dither Array element corresponds to the number of 1's that exist in the Prototype Binary Pattern, less the "1" at the location of that element.

The terms "minority pixel" and "majority pixel" are used. When less than half of the pixels in the Prototype Binary Pattern are 1's, they are the minority pixels and the majority pixels are 0's. The reverse is true when more than half of the elements are 1's. The terms "cluster" and "void" refer to the arrangement of minority pixels on the background of majority pixels. A "void" is a large space between minority pixels, and a "cluster" is tight grouping of minority pixels.

To achieve our goal of producing homogeneous distributions of 1's and 0's, minority pixels are always added in the center of the largest voids, and are removed (or replaced with a majority pixel) from the center of the tightest clusters in the Prototype Binary Pattern. The key to doing this is the filter used to find voids and clusters. This will be presented in section 3. The system that actually builds the Dither Array, presented in section 5, needs a starting point for the Prototype Binary Pattern. Thus, the Initial Binary

Pattern generator is presented first in section 4. The way in which the resulting Dither Array is normalized and used is covered in section 6, with example images presented in section 7.

3. The Void- and Cluster-Finding Filter

Since the Dither Array is periodic, the corresponding Prototype Binary Pattern will tile all of two-space as shown in figure 1. To achieve a homogenous distribution, minority pixels should be removed from clusters or inserted into voids.

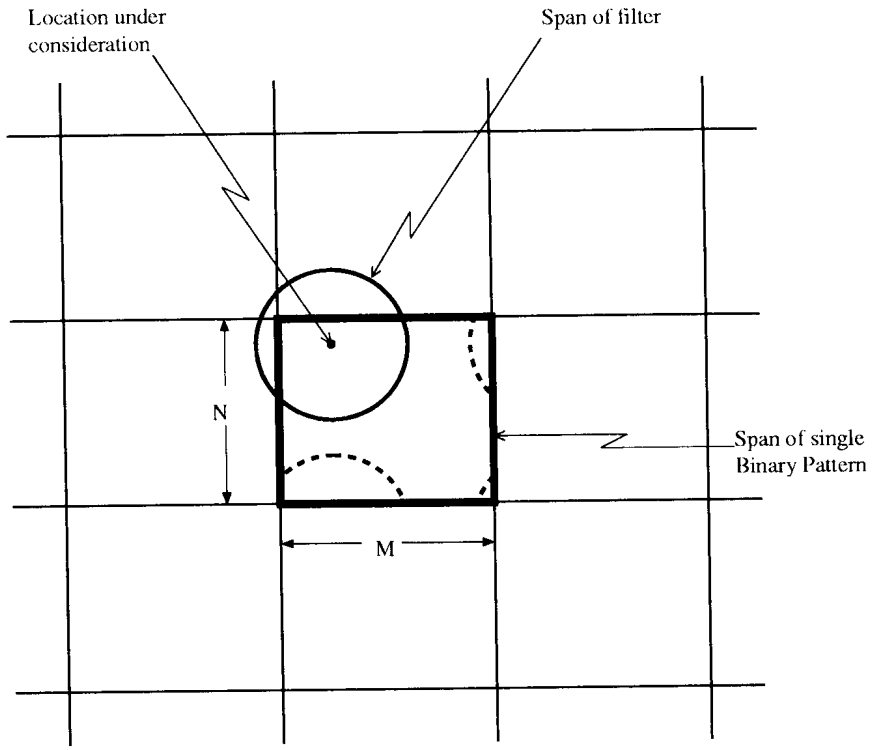

Figure 1: Two-dimensional wrap-around property. The binary pattern matrix tiles the plane, as shown.

A fundamental feature of a filter that looks for voids or clusters is the wrap-around property as represented by the coverage of the disk outlined in figure 1. When the span of such a filter extends beyond the bounds of the Prototype Binary Pattern, it must wrap around to the other side of the Prototype Binary Pattern, as indicated by the dashed lines.

A void-finding filter considers the neighborhood around every majority pixel in the Prototype Binary Pattern. A cluster-finding filter considers the neighborhood around every minority pixel. A variety of filters, both linear and nonlinear, will work.

One such filtering scheme that works well is the following form of modified convolution:

$$DA(x,y) = \sum_{p=-M/2}^{M/2} \sum_{q=-M/2}^{M/2} BP(p',q')f(p,q)$$

where

$$p' = (M + x - p) \text{ modulo } M$$
$$q' = (M + y - q) \text{ modulo } M.$$

In this equation, $DA(x,y)$ is the Dither Array, $BP(x,y)$ is the Prototype Binary Pattern, and $f(x,y)$ is the filter, or weighting function. The execution of this modified convolution equation can be made extremely efficient by taking advantage of symmetry, look-up tables, and exploiting the fact that the values of $BP(x,y)$ can only be 0 or 1.

Central to the success of this method is the choice of the filter, $f(x,y)$. A natural choice for such a filter is the two-dimensional Gaussian

$$e^{-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2}}.$$

Our experiments found that a Gaussian does indeed work well. In this paper, we will consider the important symmetric case of square pixels, where both dimensions are treated equally. This simplifies our filter to

$$e^{-\frac{r^2}{2\sigma^2}}$$

where $r^2 = x^2 + y^2$ and $\sigma = \sigma_x = \sigma_y$; r is then the distance from the location in question to the neighboring pixel.

Optimization of this method is then reduced to selecting the value for σ. Experiments consisted of generating Dither Arrays for various values of σ, dithering gray ramps with those Dither Arrays, and analyzing the quality of the patterns at all levels. Such experiments showed that the value $\sigma = 1.5$ (in terms of pixel spacing) produced the best results.

4. The Initial Binary Pattern Generator

The Prototype Binary Pattern used in the synthesis of the Dither Array needs a starting state. The method used to generate this Initial Binary Pattern is shown in figure 2. The first step is start with some input pattern, where not more than half of the pixels are 1's (and the rest 0's). This pattern can be almost anything; a white noise pattern is fine. Figure 3(a) is an example of a 16×16 input pattern that was randomly generated, with 26 minority pixels.

The purpose of the algorithm is to move minority pixels from tight clusters into large voids. With each iteration, the voids should get smaller, and the clusters looser. This is done one pixel move at a time until both the voids stop getting smaller, and the clusters stop getting looser. It turns out that the condition of convergence is quite simple; processing is complete when removing the "1" from the tightest cluster in the Prototype Binary Pattern creates the largest void.

The minority pixel (black or "1") in the tightest cluster, and the majority pixel (white or "0") in the largest void are identified in figure 3(a). After the first iteration, the black pixel with the tightest cluster is moved to the largest void, resulting in the pattern shown in figure 3(b). Once again, the locations of the new tightest cluster and new largest void are identified. It should be noted that it is entirely possible for minority pixels to be moved more than once; the search for voids and clusters at each iteration is independent of past moves.

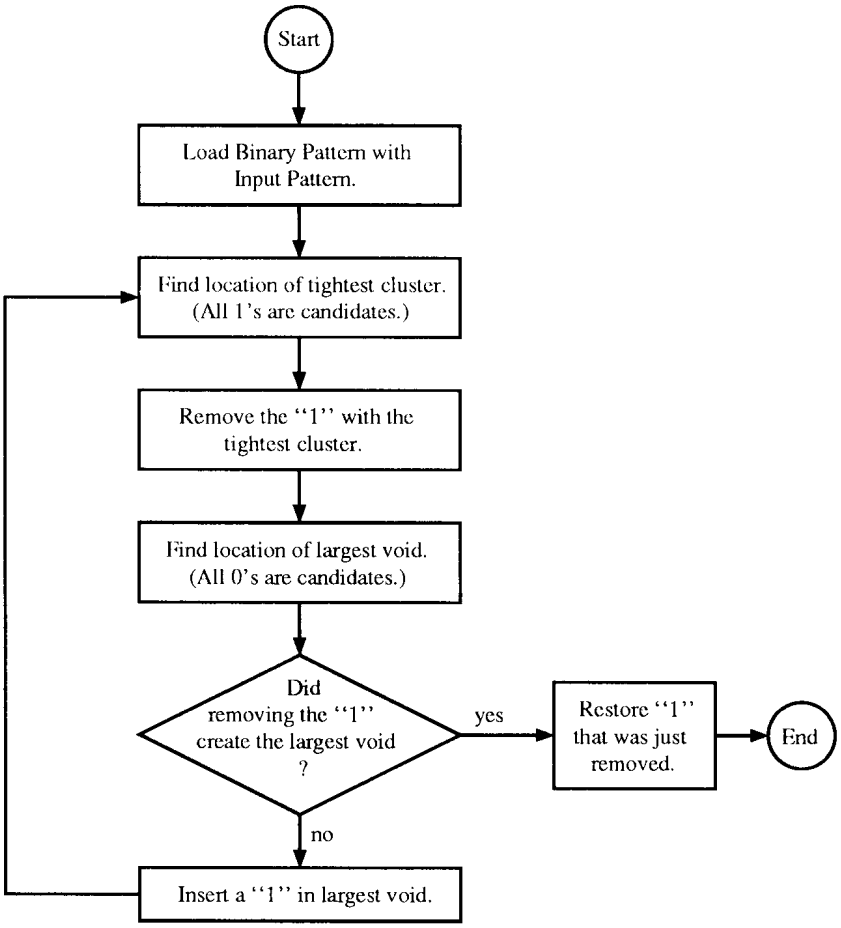

Figure 2: Initial Binary Pattern Generator.

The process continues until it converges. The results of this example are summarized in figure 4 where 12 iterations were needed. Four periods are shown of both the (a) input pattern, and (b) the rearranged or "Initial Binary Pattern", to illustrate the wrap-around or edge-abutting consequences of tiling two-space with such patterns. Note how homogeneously distributed the resulting pattern is.

5. The Dither Array Generator

With this starting point, the Initial Binary Pattern, the Dither Array can now be built. The Dither Array is generated in 3 phases corresponding to the range of *Rank* values that are entered. Using the constant *Ones* to represent the number of 1's in the Initial Binary Pattern, Phase I enters *Rank* values between *Ones* and 0 into the Dither Array. Phase II enters values between *Ones* and the half-way point, and Phase III enters values from the half-way point to all 1's. As mentioned earlier, the Prototype Binary Pattern and Dither Array are manipulated in parallel.

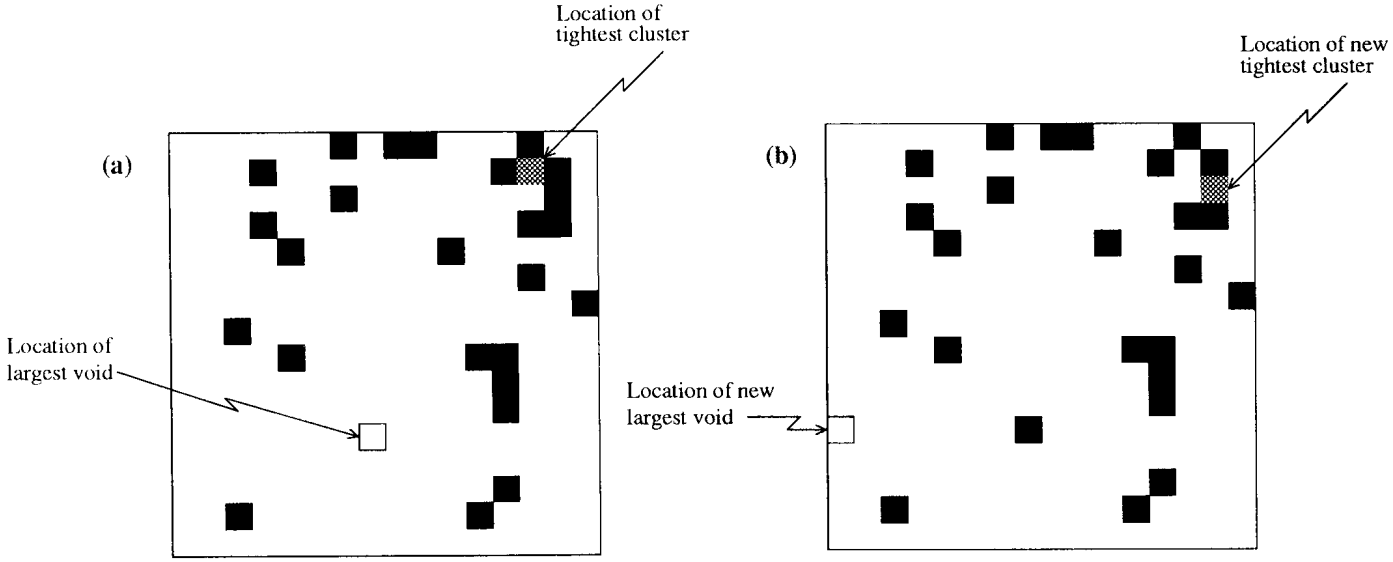

Figure 3: Example of the first two iterations of the Initial Binary Pattern Generator. The 16x16 Input pattern in (a) has 26 minority pixels (1's).

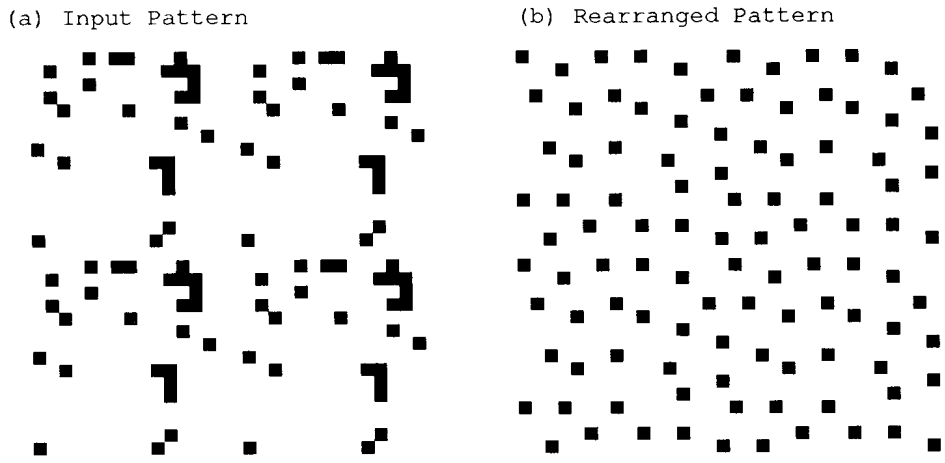

Figure 4: Results of the Initial Binary Pattern Generator. Four periods of the 16x16 Input pattern (a) and the Rearranged pattern (b) are shown to illustrate the wrap-around properties. The Input pattern is the same as that shown in figure 3(a).

A flow diagram of the Dither Array Generator is depicted in Figure 5, with Phase I in part (a), Phase II in part (b) and Phase III in part (c).

We start by loading the Prototype Binary Pattern with the Initial Binary Pattern in Phase I (figure 5(a)). One minority pixel, or "1", is removed at a time, while in parallel the *Rank* is entered in the Dither Array in the exact corresponding location as the "1" in the Prototype Binary Pattern just removed. In each case, the minority pixel that is removed is the one identified as that in the tightest cluster. The *Rank* entered in the Dither Array is equal to the number of 1's left in the Prototype Binary Pattern just after the current one is removed. So, in Phase I, the *Rank* values entered in the Dither Array start with one less than the number of 1's in the Initial Prototype Binary Pattern, and reduce one at a time to 0. The Prototype Binary Pattern at the end of Phase I is a pattern with all 0's.

In Phase II (figure 5(b)), we begin again by loading the Prototype Binary Pattern with the Initial Binary Pattern. In this case one minority pixel or "1" is inserted at a time, while in parallel, the *Rank* is entered in the Dither Array. In each case, the minority pixel that is inserted is done so at the location of a "0" identified as the center of the largest void. The *Rank* entered in the Dither Array is equal to the number of 1's in the Prototype Binary Pattern just before the current one is inserted. In Phase II, the *Rank* values entered in the Dither Array start with the number of 1's in the Initial Binary Pattern and increase one at time to $\text{int}\{\frac{MN}{2}\}$. The Prototype Binary Pattern at the end of Phase II has an equal number of 1's and 0's, except the case where M and N are both odd where there will be one more "1" than 0's.

Phase III (figure 5(c)) starts with this Prototype Binary Pattern. It is important to note that in this phase, the meaning of "minority pixel" is reversed from "1" to "0"; that is, there will be more 1's than 0's. The Prototype Binary Pattern is filled with more and more 1's, while the Dither Array continues to be filled with values of *Rank* increasing from $\text{int}\{\frac{MN}{2}+1\}$ to MN. In each step, a "1" is inserted in the "0" location identified in the tightest cluster of minority pixels (now 0's).

At the end of Phase III, the Prototype Binary Pattern is filled with all 1's, and the Dither Array is fill with a unique value of *Rank* in each element, from 0 to $(MN-1)$.

6. Normalization

The Dither Array is now ready to be used for either bitonal or multilevel dithering. In general the input image signal will have NIL (number of input levels) discrete values, and the desired output signal will have NOL (number of output levels) discrete values. The details of an efficient implementation of multilevel dithering that allows any value of NIL and NOL (where NOL is less than or equal to NIL) is given in [10]. A key property of this method is the resulting dithered average output is always equal to the input. The expressions for proper normalization is summarized here.

We define a Quantizer with equal steps of size

$$\Delta_Q = \frac{NIL-1}{NOL-1}.$$

The elements of the Dither Array are normalized by means of the following relation:

$$Rank' = \text{int}\left\{\frac{\Delta_Q}{MN}(Rank+\frac{1}{2})\right\}$$

where "int{}" represents integer truncation, or the rounding-down operation.

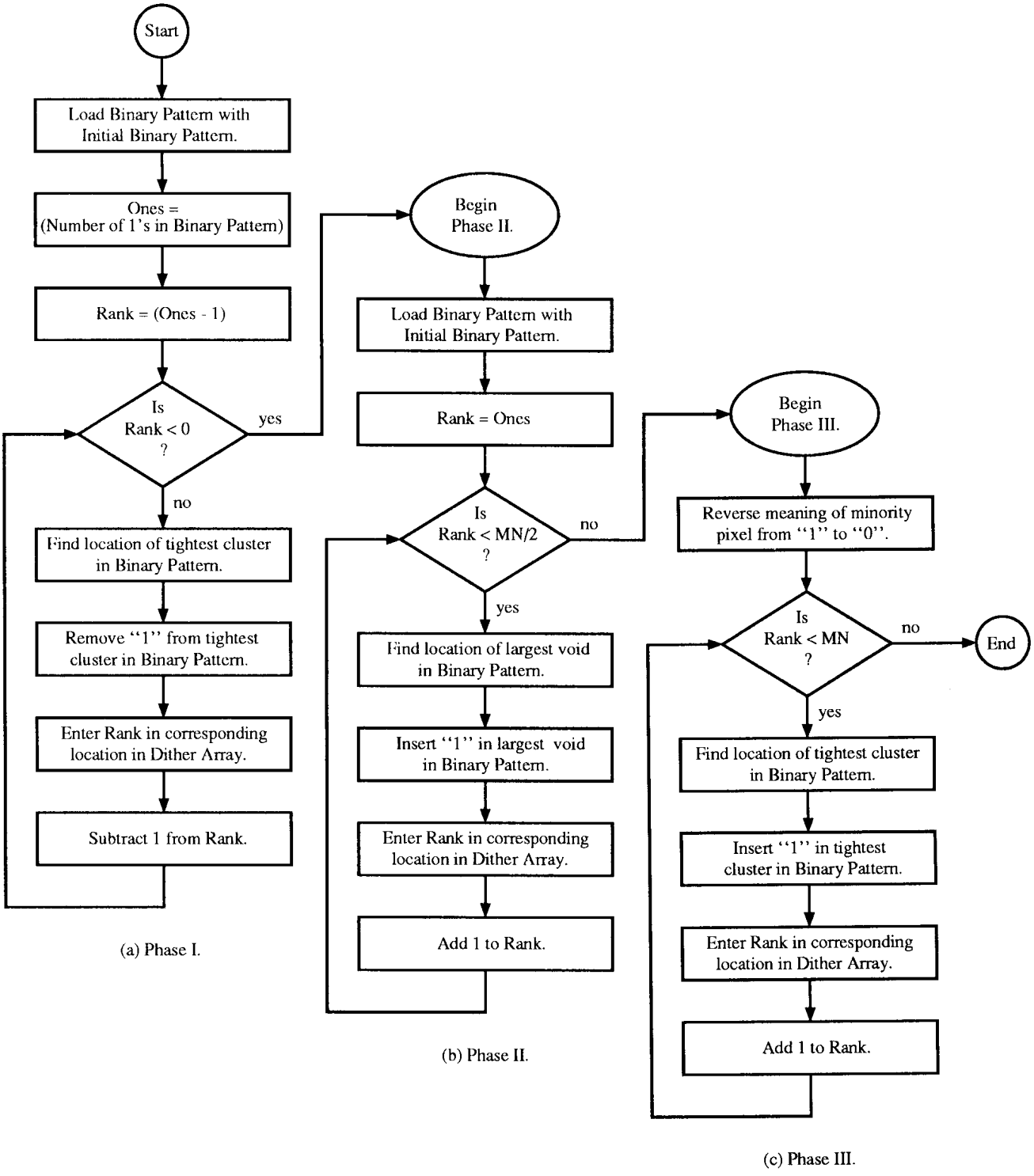

Figure 5: Dither Array Generator. (a) Phase I: Elements form (*Ones*-1) to 0 are entered. (b) Phase II: Elements form *Ones* to $(\frac{MN}{2} - 1)$ are entered. (c) Phase III: Elements from $(\frac{MN}{2})$ to MN are entered.

For a given *input*, the *output* is then computed as

$$output = \text{Quantizer}\{input + Rank'\}.$$

The value $Rank'$ is the element of the Dither Array with the same x and y address, or index, as *input* and *output*, modularly adjusted by M and N. That is, for a given image address (x, y), the index to the Dither Array is

$$(x \text{ modulo } M, y \text{ modulo } N).$$

For the important special case of bitonal output, that is when $NOL = 2$, the computation can be simplified as

$$\text{if}(input > Rank') \text{ then } output = 1 \text{ else } output = 0.$$

This method of dithering will produce a uniformly spaced distribution of dithered levels, that preserves the average value of the input.

7. Examples

While a particular method may perform well for some select gray levels, it may fail terribly for others. For this reason, an important assessment mechanism for any dithering process is a visual evaluation of the rendering of all gray levels. The result of dithering a gray ramp with a 32 by 32 array generated by the void-and-cluster method described in this paper using a Gaussian filter with $\sigma = 1.5$ is shown in figure 6(c).

For comparison, the results of dithering by two other means are shown as well. In figure 6(a), the ramp is dithered with a 16 by 16 recursive tessellation ordered dither array. Unlike (a) and (c), the ramp shown in figure 6(b) was not rendered by ordered dither. Instead, it was the result of the error diffusion method using perturbed Floyd and Steinberg weights, and processed on a serpentine raster, as described in [11].

The patterns in the ramp in (c) do not suffer the strong horizontal and vertical structures found in (a), and are more consistently homogeneous than those found in (b).

The same three methods were used to render a "standard" scanned image in figure 7. All images in these examples are printed at 100 dpi. The same pattern characteristics are present here as in figure 6. The error-diffusion-based method in figure 7(b) appears sharper than the other two. It is well known that error diffusion has inherent sharpening characteristics, and the impulse response of the effective sharpening filter has in fact been measured for the case of a horizontal step function [12].

It is also known that the inherent sharpening in error diffusion is not isotropic. It may therefore be better to control sharpening with a filtering stage independent of dithering, with a Laplacian filter for example.

8. Concluding Remarks

A new method for generating dispersed-dot ordered dither arrays was presented. The algorithm is quite simple and produces excellent results with relatively small array sizes.

The void-and-cluster method is also very general. For the special case where M and N are equal powers of 2, and the Initial Binary Pattern is chosen to be one with a single "1" pixel, the resulting dither array is precisely what would result from the method of recursive tessellation! For a 16 by 16 array size the result would be an array equivalent to the one used to produce figures 6(a) and 7(a). In fact the same array results when the Initial Binary Pattern is *any* recursive tessellation pattern of a fixed gray level.

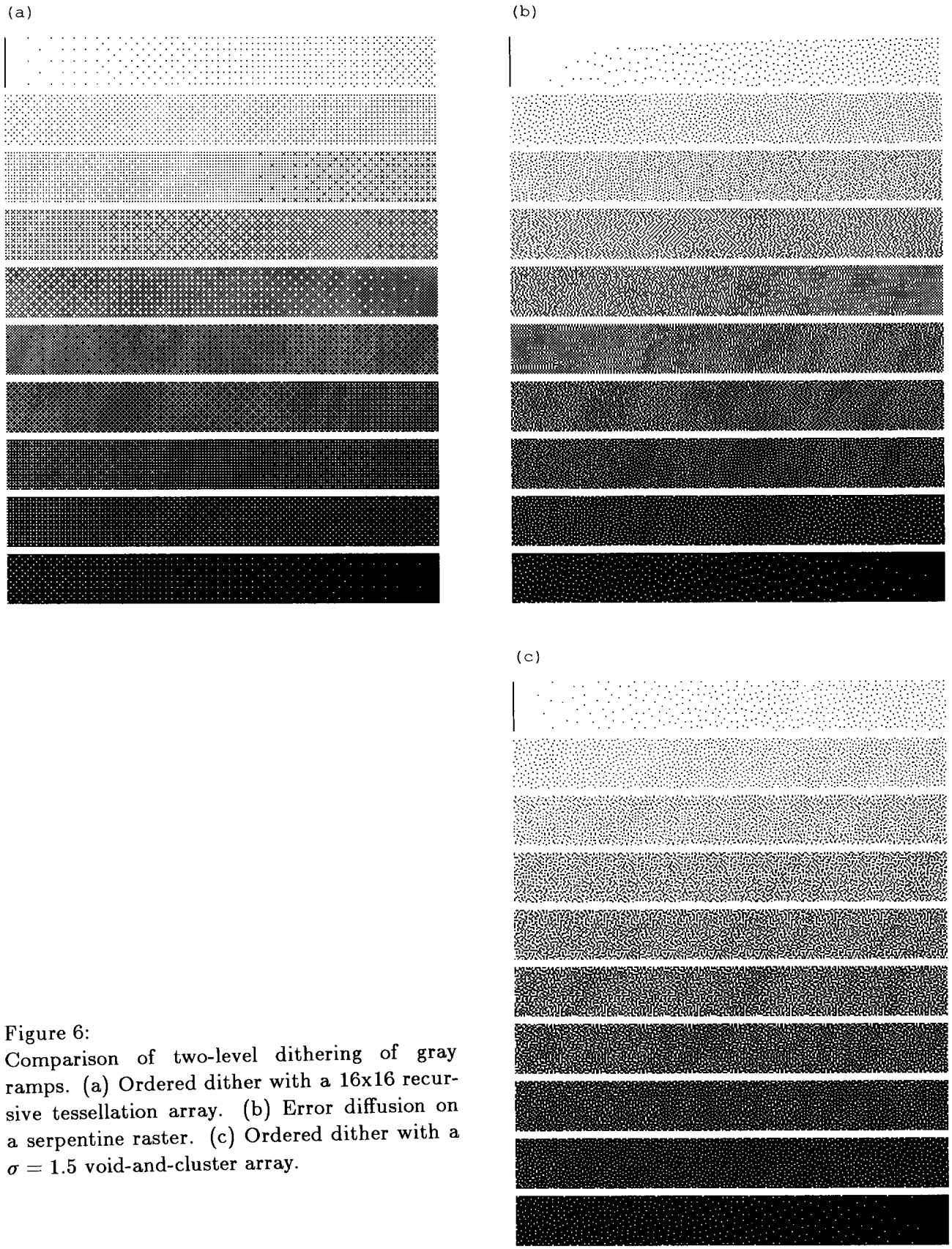

Figure 6:
Comparison of two-level dithering of gray ramps. (a) Ordered dither with a 16x16 recursive tessellation array. (b) Error diffusion on a serpentine raster. (c) Ordered dither with a $\sigma = 1.5$ void-and-cluster array.

(a)

(b)

(c)

Figure 7:
Comparison of two-level dithering of a scanned image. (a) Ordered dither with a 16x16 recursive tessellation array. (b) Error diffusion on a serpentine raster. (c) Ordered dither with a $\sigma = 1.5$ void-and-cluster array.

While not explored here, the void-and-cluster method will solve the problem of dithering on asymmetric rectangular grids, that is, on grids where the horizontal sample period is not equal to the vertical sample period. In such a case, the void-finding and cluster-finding filter discussed in section 3 simply needs to account for the different spacings in the two dimensions. Also, hexagonal void-finding and cluster-finding filters can also be used for generating Dither Arrays for hexagonally sampled data.

9. References

[1] J.C. Stoffel and J.F. Moreland, "A survey of electronic techniques for pictorial reproduction", *IEEE Tran. Commun.*, vol. 29, pp. 1898-1925, 1981.

[2] J.F. Jarvis, C.N. Judice, and W.H. 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.

[3] R. Ulichney, *Digital Halftoning*, The MIT Press, Cambridge, 1987.

[4] B.E. Bayer, "An optimum method for two level rendition of continuous-tone pictures", *Proc. IEEE Int. Conf. Commun., Conference Record*, pp. (26-11)-(26-15), 1973.

[5] R. Ulichney, "Frequency Analysis of Ordered Dither," *Proc. SPIE, Hard Copy Output*, vol. 1079, pp. 361-373, 1989.

[6] J.P. Allebach, and B. Liu "Random quasi-periodic halftone process", *J. Opt. Soc. Am.*, vol. 66, pp. 909-917, 1976.

[7] J.P. Allebach "Random nucleated halftone screening", *Photogr. Sci. Eng.*, vol. 22, no. 2, pp. 89-91, 1978.

[8] T. Mitsa and K. Parker "Digital halftoning using a blue noise mask", *Proc. SPIE, Image Proc. Algorithms and Techniques II*, vol. 1452, pp. 47-56, Feb 25 - Mar 1, 1991.

[9] K. Parker, T. Mitsa, and R. Ulichney "A new algorithm for manipulating the power spectrum of halftone patterns", *SPSE's 7th Int. Congress on Non-Impact Printing*, Portland, OR, pp. 471-475, Oct. 10, 1991.

[10] R. Ulichney, "Video Rendering", *Digital Technical Journal*, vol. 5, no. 2, 1993.

[11] R. Ulichney, "Dithering with blue noise", *Proc. IEEE*, vol. 76, no. 1, pp. 56-79, 1988.

[12] K.T. Knox, "Edge Enhancement in error diffusion", *SPSE's 42nd Annual Conf.*, pp. 310-313, May 14-19, 1989.