One-dimensional Dithering

Robert Ulichney, Cambridge Research Lab,
Digital Equipment Corp, One Kendall Sq., Cambridge, MA 02139, USA[†]

ABSTRACT

A real-time inverse dithering system for video display can be implemented very efficiently if operations are needed on only the current scan line. To optimize overall display quality, a corresponding one-dimensional ordered dither array is sought. This paper describes a one-dimensional recursive tessellation algorithm. A serendipitous implementation involves a simple bit-reversal of the horizontal pixel address. To optimize two-dimensional homogeneity, the 1-D array is phase adjusted in the vertical direction. A scheme for selecting candidate phase vectors is also presented. The recursive tessellation algorithm is generalized to identify equivalence class arrays that share the same homogeneity property but have different ordering.

Keywords: dithering, halftoning, inverse dithering, ordered dither

1. INTRODUCTION

Among the many classes of techniques for dithering an image, the simplest to implement are those described as ordered dither. Ordered dither is a point process where computation uses only the current pixel in the input image to generate the corresponding pixel in the output image; no surrounding pixels are used. The input is quantized based on a dither value from a deterministic, periodic dither array. The advantage of ordered dither is that is allows for simple and fast implementations. Key to the quality of the resulting output image is the dither array. One popular method of order dither in two-dimensions is know as recursive tessellation [1, ch. 6]. This type of dither is also known as Beyer's dither [2].

An important special case is where the number of input and output levels to the dither system is a power of two. In this case the dither array with 2^N values numbered 0 to (2^N-1) can be used as is without further normalization given that the other elements in the dither system are properly designed. In particular, the image raw input values must be adjusted. An overview of how this can be achieved is presented in [3].

Figure 1 shows one situation where dithering is used. An input image is dithered to reduce video frame-buffer memory size. To improve the final display quality, an inverse-dithering step can be used to reconstruct as closely as possible the original digital image before displaying it on a monitor. The goal of inverse dithering is to reconstruct from the dithered image the original image with a larger pixel bit-depth. Several approaches have been suggested, including wavelet decomposition [4], MAP estimation [5], and adaptive algorithms [6][7].

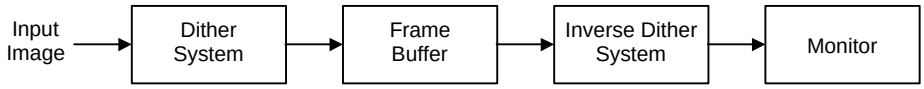

Figure 1. Dithering and Inverse-dithering systems to improve image quality.

We designed a video-rate hardware inverse dithering system with a premium on chip space. Experiments demonstrated that very good results are achievable when processing was limited to a single image line. Such a scheme has tremendous implementation advantages since line buffers are not needed; the algorithm operates only on the current image line without any regard for image data before or after that line. When a priori knowledge of the dither array is exploited, our tests showed that reconstruction of the original image is indeed improved.

For such a one-dimensional scheme to work best, it is desirable that the same dither array be used for every line. A one-dimensional dither array would be ideal for this application. Other studies of 1-D dithering include more complex neighborhood-operation-based schemes such as in a heuristic study of error-diffusion [8] or model-based halftoning [9]. Our focus, and focus of this paper, is on the design of a 1-D ordered dither system, and the effective 2-D system that incorporates it.

[†] For further author information –
WWW: http://www.crl.research.digital.com/who/people/ulichney/bio.htm; Email: ulichney@crl.dec.com

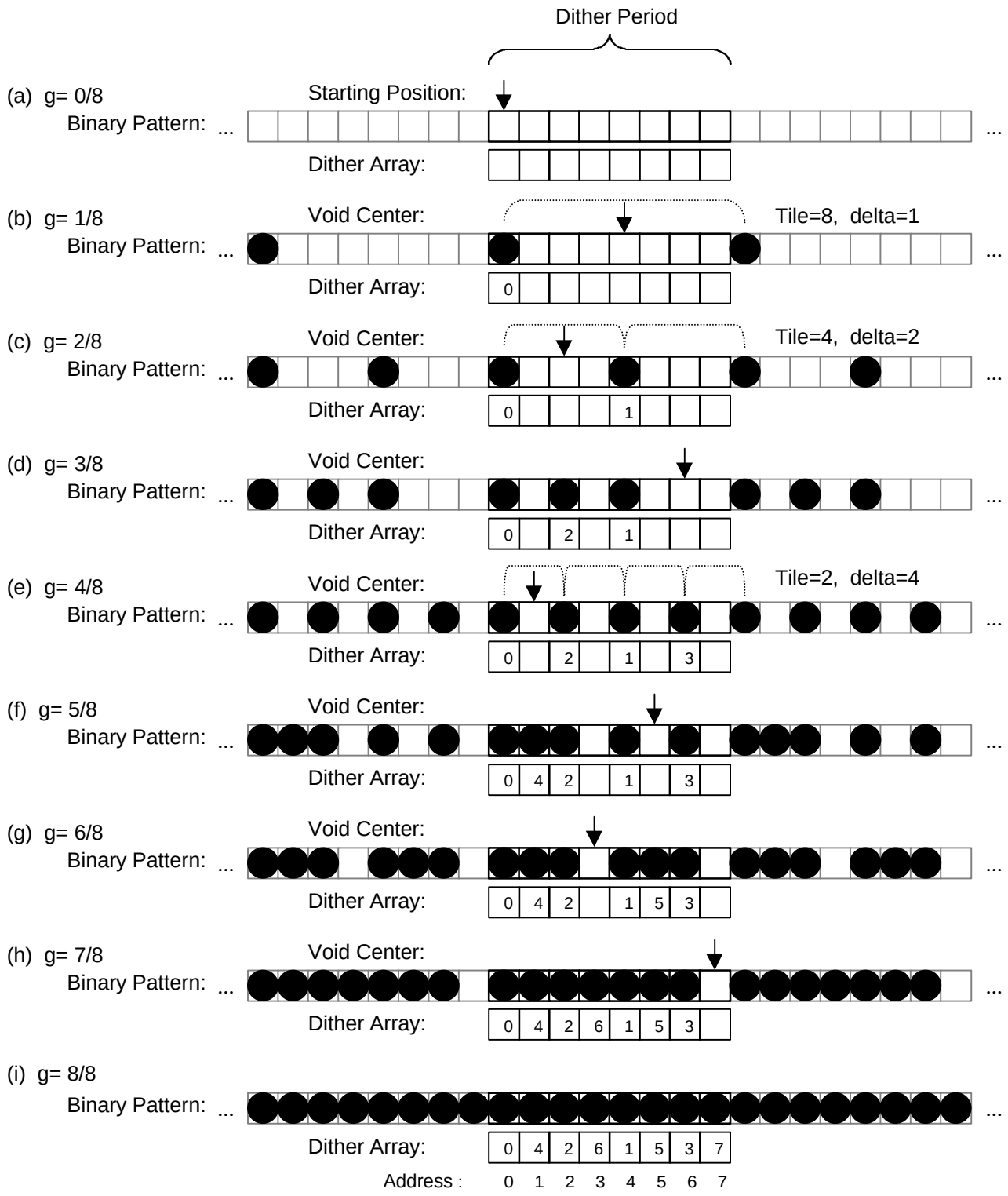

Figure 2. One-dimensional recursive tessellation.

2. 1-D RECURSIVE TESSELLATION

As was described above, the dither array of interest is an ordered array of integers from 0 to (2^N-1), where 2^N is the number of array levels. While dither arrays can be used in multilevel systems, it is easiest to design and illustrate the generation of the array for a bitonal system where values can only be black or white.

For a dither array with 2^N elements, we consider the problem of representing (2^N+1) fixed flat gray levels from full white (0 elements on) to full black (2^N elements on), and all the integer steps in-between. The goal is to produce a dither array whose members are ordered as homogeneously as possible keeping in mind that the array is to be repeated periodically; thus the beginning of the array effectively wraps around to the end of the array.

Figure 2 illustrates the step-by-step design process for the case of an 8-element dither array. The figure shows the build-up of the binary pattern from (a) 0 out of 8 pixels on, to (i) 8 out of 8 pixels on. The binary pattern is assumed to extend forever to the left and to the right replicating the same 8-pixel pattern, the fundamental period of which is indicated on the top of the figure.

The order that pixels are turned on is governed by the dither array. The state of build-up of the dither array is also shown in each step in. Figure 2. At the bottom of the figure, the address of the elements in the dither array is indicated. . The mission is to order the 8-element array from 0 to 7 in such a way as to make the resulting binary pattern as homogeneous as possible for all gray levels.

The algorithm begins in part (a) with a blank binary pattern and dither array. The arrow indicates the selection of the first location as the beginning of the period. In Figure 2(b) the selection from part (a) is manifested as a black pixel in the binary pattern and value of 0 in the dither array. This results in a "tile" of size 8 pixels as indicated, periodically replicated. To maximize homogeneity the next pixel to turn on should be in the center of the void between with "on" pixels. The arrow in part (b) selects the center of this void, and the center of the tile.

In part (c) this choice is shown as black pixel in the selected location and a value of 1 in the corresponding location in the dither array. A new periodic tile of size 4 is evident. The center of the first void or tile is indicated with an arrow as the next selection. Part (d) shows this choice with a value of 2 in the dither array. Also shown in part (d) is the center of the remaining void as the next selection.

There is a new tile size in force of 2 pixels in Figure 2(e). 4 tiles of size 2 are shown, with an arrow in the center of the first one indicating the next selection. In parts (b) (c) and (e) a value of "delta" is shown. In each case the value of the dither array entry that corresponds to the center of a particular tile happens to simply be the value of the already-assigned dither array element at the right edge of that tile added to the value of delta. This is true for the remaining selections in parts (f) through (i). The finished dither array for this 8-element case is shown in Figure 2(i).

The number of one-dimensional tiles sizes that are recursively tessellated in this 8-element case was 3. In the general case of dither periods of size 2^N there are N tiles of size 2^N, 2^{N-1},…, and 2^1.

An algorithm for generating a 1-D dither array given a power, N, is shown in Figure 3. This system in this figure builds an array "Dither" of size 2^N. The process begins by setting the first dither array element, Dither[0]=0, and initializing a counter b=0. The top-level loop has N steps, one for each tile size. In each of these steps, the tile is set to size 2^{N-b}, and delta is set to 2^b. The actual assignment of dither array values occurs in a nested loop with 2^b steps; one step for each of the tiles in the dither period where the address at the center of each tile is assigned a value. That value is equal to the already-assigned dither array value at the beginning of the tile plus the value of delta.

Upon evaluation of the algorithm in Figure 3 it can be shown that the values of the resulting dither array will always be the bit reverse of the address of those values. In the top-level N-step loop, the counter b represents the bit that is being assigned in the bit-reversed value. The adding of the value delta in the lowest box of Figure 3 has the effect of setting the b^{th} bit. (It is interesting to note this algorithmic flow can in fact be a very efficient means for generating a bit-reversed table. If the algorithm of Figure 3 is optimized to use shifts instead of exponentiation, it has less steps than the popular reverse-bit counter algorithm used in the FFT [10].)

So, the key observation is that the values of our homogeneous 1-D dither arrays are always available as simply the bit reverse of the pixel address! Indeed this is true for the 8-element case illustrated in Figure 2, and is as shown in Table 1.

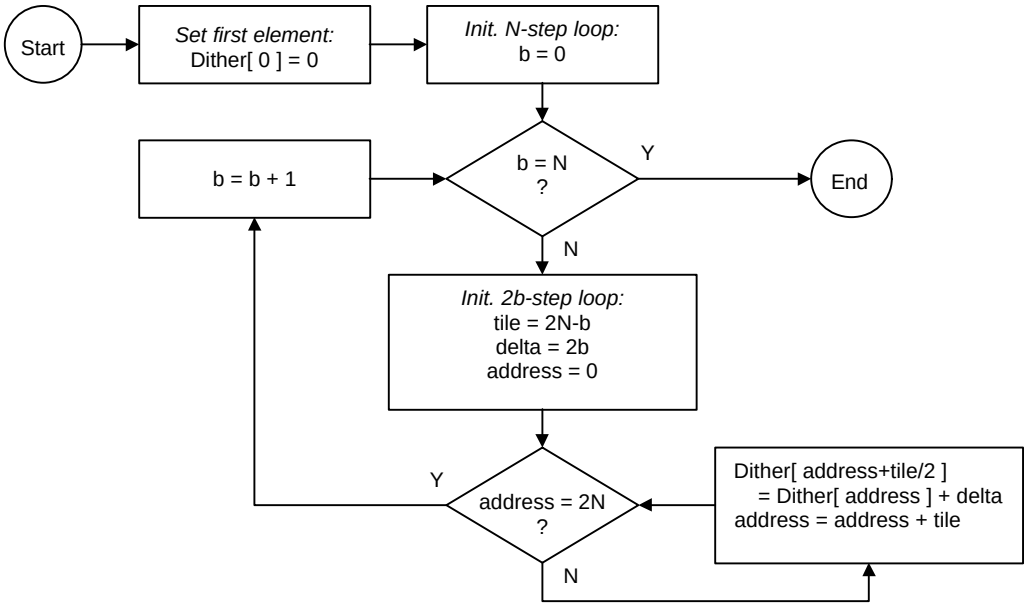

Figure 3. 1-D recursive tessellation algorithm.

In hardware, implementing a bit reversal is nothing more than wiring. Given an M-bit horizontal pixel address of an image, if an N-bit dither array were used where N<M, the lower N bits of the resulting address would be used to form the dither value. This is realized by simply reversing the significance of those N bits; the least significant bit of the address would serve as the most significant bit of the dither value.

Address		Value	
Decimal	binary	binary	decimal
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Table 1. Decimal and binary addresses and values for an 8-element dither array.

3. 2-D DITHER WITH A 1-D DITHER ARRAY

If the same initial phase where used for all lines in an image, the 1-D dither patters would be copied on every line resulting in vertical streaks. This is illustrated in Figure 4 for the case of an 8-element 1-D dither array. Fixed gray levels with g=0/8, 1/8, 2/8, 3/8, 4/8, 5/8, 6/8, 7/8, and 8/8 are dithered. For each gray level, a 16x16 bitonal pixel image is shown. While the patterns may be homogenous in the horizontal direction, this figure clearly illustrates the lack of homogeneity in two dimensions.

Figure 4. 8-element 1-D dither patterns using the same initial phase for each line.

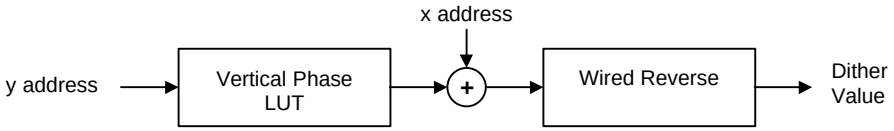

Figure 5. Random access 2-D dither system.

Of course, the fundamental characteristics of the one-dimensional dither array described here do not change by phase shifting it. A 2-D dither system should exploit this initial phase feature as shown in Figure 5. A set of ordered vertical N-bit phases are stored in a Vertical Phase Look Up Table (LUT). The number of phases is chosen to be equal to the number dither array values, so that the vertical period is equal to the horizontal period. The size of the Vertical Phase LUT, and thus the number of phases, could in fact be any value, but is set to 2^N to allow for one instance of each possible phase 0 through (2^N-1), and imparts some symmetry in the effective distribution of the resulting dither patterns.

The least significant N bits of the y address are used to address the Vertical Phase LUT. The LUT output is added to the least significant N bits of the x address. The resulting dither array value is, produced by wire reversing the output of the adder.

4. DESIGN OF THE VERTICAL PHASE ARRAY

The quality of the 2-D dither patterns that result hinge on the nature of the vertical phases. We will focus on solutions for the Vertical Phase Array of size 2^N where each value 0, 1, …, (2^N-1) is included once and only once in the array. The number of permutations of the set of numbers $\{0, 1, \ldots, (2^N-1)\}$ is (2^N)-factorial.

This can be a very large number. For a 32x32 dither array, $32! = 2.6\text{x}10^{35}$ candidate solutions. Choosing the best one requires analyzing and quantifying the quality of all of the dither patterns that result from each candidate. If some hyper-computer could analyze a billion candidates a second, and if a billion such systems could be run in parallel, it would still take the approximate age of the universe to assess all of them.

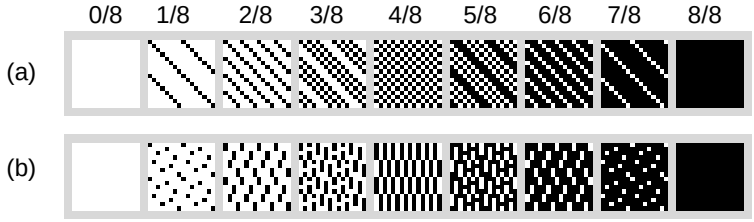

Figure 6. 8x8 element dither patterns using phase array (a) [0 1 2 3 4 5 6 7], and (b) [0 4 2 6 1 5 3 7].

Candidates that lack homogeneity should be ruled out. One such 8x8 element case is illustrated in Figure 6(a) where the vertical phase array was [0 1 2 3 4 5 6 7]. Since homogeneity is desirable, an obvious choice is to use the bit-reversed 1-D dither array. For the 8x8 element case, the phase array would be [0 4 2 6 1 5 3 7]. This is shown in Figure 6(b). While the output here is much more uniformly distributed, there is still considerable clumping of like-colored pixels.

Since homogeneity is sought, the design of the vertical phase array is much like designing a 1-D dither array. Indeed the system of Figure 5 is really the cascade of two 1-D dither arrays.

4.1 Choices in Building a 1-D Dither Array

It appears that the homogeneous nature of the bit-reversed 1-D dither array is a good choice for a vertical phase array, but having the identical order as the horizontal dither array, it interacts in such as way as to form some correlated clumping. In this section the 1-D recursive tessellation algorithm is re-evaluated to find equivalently homogeneous distributions but with different dither arrays. Then the resulting dither arrays can be used as candidate vertical phase arrays.

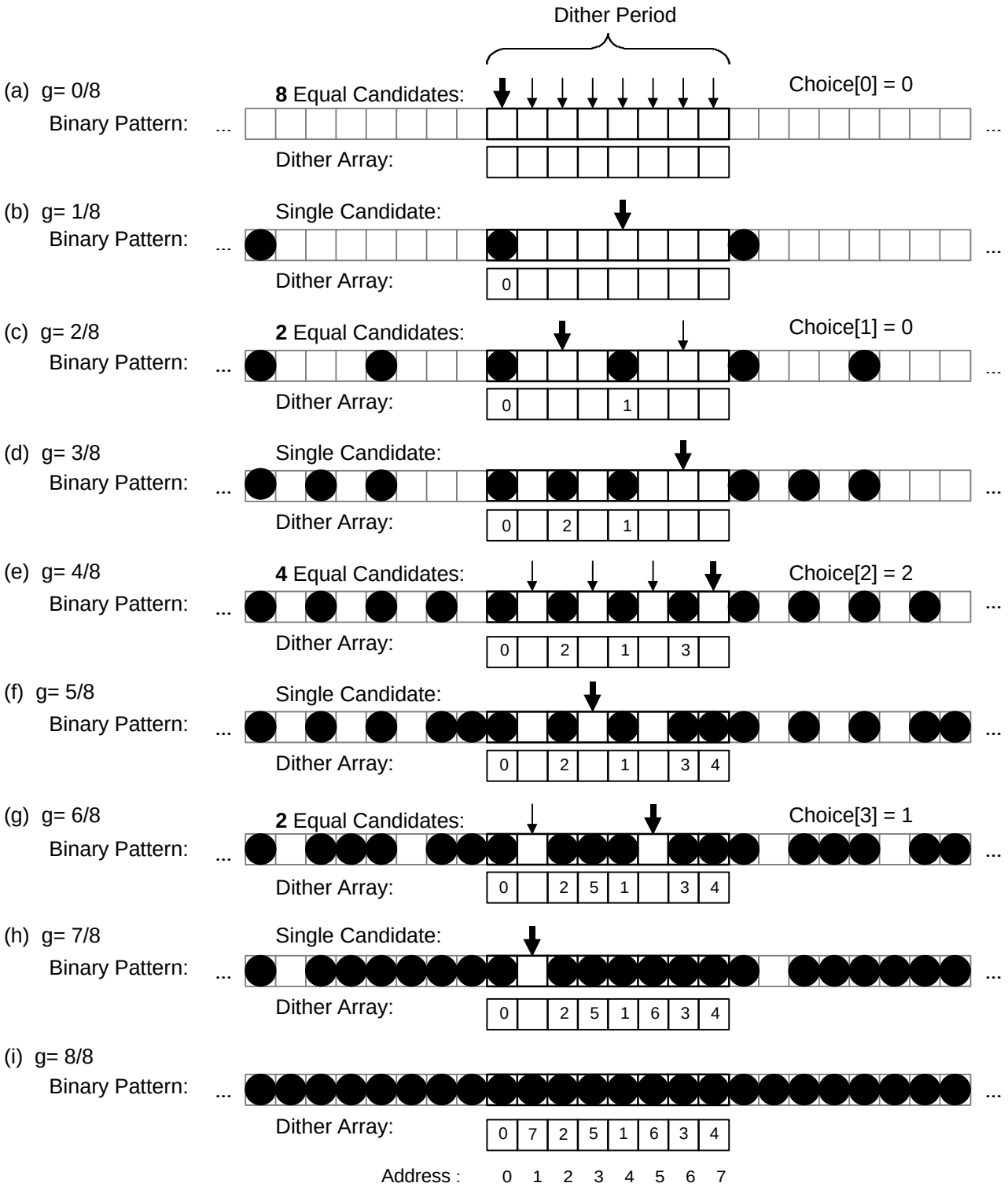

Figure 7. Process for building a homogeneous vertical phase array.

Figure 7 illustrates the step-by-step design process for the case of an 8-element dither array. The figure shows the build-up of the binary pattern from (a) 0 out of 8 pixels on, to (i) 8 out of 8 pixels on. The binary pattern is assumed to extend forever to the left and to the right replicating the same 8-pixel pattern, the fundamental period of which is indicated on the top of the figure.

The order that pixels are turned on is governed by the dither array. The state of build-up of the dither array is also shown in each step in Figure 7. At the bottom of the figure, the address of the elements in the dither array is indicated. . The mission is to order the 8-element array from 0 to 7 in such a way as to make the resulting binary pattern as homogeneous as possible for all gray levels.

The algorithm begins in part (a) with a blank binary pattern and dither array. In this and in other parts, arrows indicate candidate choices that will produce equally homogeneous distributions. In part (a) all 8 locations are equal possibilities. Also in this and in other parts of Figure 7, the candidate that is chosen in indicated by a bold arrow. In part (a) the 0^{th} location is selected.

In Figure 7 (b) the selection from part (a) is manifested as a black pixel in the binary pattern and value of 0 in the dither array. This results in a "tile" of size 8 pixels periodically replicated. To maximize homogeneity the next pixel to turn on should be in the center of the void between with "on" pixels. The arrow in part (b) selects the center of this void, and the center of the tile. There are no other candidates equal in homogeneity to this one, and so the choice is unconditionally the single candidate shown.

In part (c) this choice is shown as black pixel in the selected location and a value of 1 in the corresponding location in the dither array. A new periodic tile of size 4 is evident. In this case there are 2 equally likely candidates –the centers of the two tiles. Either one can be chosen, and in this example the first is selected as indicated by the bold arrow. Part (d) shows this choice with a value of 2 in the dither array. Also shown in part (d) is the center of the remaining void as the single and unconditional next selection.

There is a new tile size in force of 2 pixels in Figure 7(e). 4 tiles of size 2 are shown, with arrows marking the void centers of each of the 4 equal candidates. These are shown at addresses 1, 3, 5, and 7. We adopt the convention of counting candidates from left to right starting with 0 from the location of the last choice. The previous part with a choice was (c) where the choice was address 2. So, choice 0 would be address 3, choice 1 would be address 5, choice 2 would be address 7, and choice 3 would be address 1. In part (e) the choice is 2, the location at address 7 and is indicated with a bold arrow.

In Figure 7(f) once again we have a single unconditional choice where only one location is the most homogeneous candidate. In general, every other step will have an unconditional choice that is exactly one half dither period away from the last selection. In part (g) the remaining two candidates are equally likely. The location at address 1 is choice 0 and the location at address 5 is choice 1. In this example the choice taken is 1.

The final and unconditional choice is shown in Figure 7(h). In part (i) the final dither array is shown: [0 7 2 5 1 6 3 4].

4.2 Generalized 1-D Recursive Tessellation

This process can be generalized for any power of N. For an array with size = 2^N, exactly half of those elements have a greater-than-one-choice of candidates. In the 8 element array of Figure 7, 4 values had equally homogeneous choices: Part (a) had 8 choices, part (c) had 2 choices, part (e) had 4 choices, and part (g) had 2 choices. We would then say that the choice size vector for an array of size 8 was [8 2 4 2]. The choice size vectors for arrays of sizes 4, 8, 16, 32, and 64 are shown in Table 2.

Size	Choice Size Vector	Number of Unique Choices	Total Unique Arrays
4	4, 2	0	1
8	8, 2, 4, 2	2	8
16	16, 2, 4, 2, 8, 2, 4, 2	6	1024
32	32, 2, 4, 2, 8, 2, 4, 2, 16, 2, 4, 2, 8, 2, 4, 2	14	33554432
64	64, 2, 4, 2, 8, 2, 4, 2, 16, 2, 4, 2, 8, 2, 4, 2, 32, 2, 4, 2, 8, 2, 4, 2, 16, 2, 4, 2, 8, 2, 4, 2	30	7.205794×10^{16}

Table 2. Choice Size Vectors for various array sizes.

The product of all of the elements in the choice size vector equals the total number of dither arrays that are equally homogeneous as the bit-reverse array. However, many of them differ by a translational shift or mirror reflection. The translational shifted versions can be omitted by removing the first element of the choice size vector. That is, force the first element to be selected to always be the first location. In Figure 7(a) that would eliminate the 8 choices. Mirror reflected arrays could be omitted by throwing away the second element of the choice size vector.

In Table 2 the column labeled "Number of Unique Choices" is the number of elements in the choice size vector less one for translational redundancy and one for mirror redundancy. "Total Unique Arrays" is the product of the unique choice sizes. The important point is that these totals are much less than (2^N)-factorial. Of particular interest is for size=32, the number of permutation candidates dropped from $\sim 3x10^{35}$ to the much more manageable $\sim 3x10^{7}$.

Based on this generalize process, a system for building a vertical phase array is shown in Figure 8. The size of the Phase array is set to equal 2^N. A key input to the Phase Array Generator is the selection of choices in the "Choices" vector. There are size/2 elements in the ChoiceSizes vector and the Choices vector. The value of each element in the Choices vector must be less than the corresponding element in the ChoiceSizes vector. The values of the ChoiceSizes vector are as stored in Table 2.

With this initialization information, the Phase Array Generator produces a unique Phase Array with a number of elements equal to "size". The algorithm of Figure 8 generalizes the process shown in Figure 7. "rank" is the value of elements in the phase array. It starts with value 0 and counts up to value (size-1). The computation of the addresses into which the value of rank is loaded is the focus of the algorithm.

For each consecutive pairs of ranks (phase array values), one case has a Choice associated with it and the other is assigned an unconditional position. Pairs of ranks are marked by the index "i". For each pair, a step size is calculated, that is equal to the tile size in 1-D Recursive tessellation algorithm of Section 2. An offset is found that is equal to half of a step. The address is found to be equal to the last address plus the offset plus Choice steps. Since the sum may be greater than "size", it is normalized by modulo size (shown at "mod size" in Figure 8). The 2nd value of a pair of ranks is always assigned to the unconditional address that is size/2 units away from the last rank.

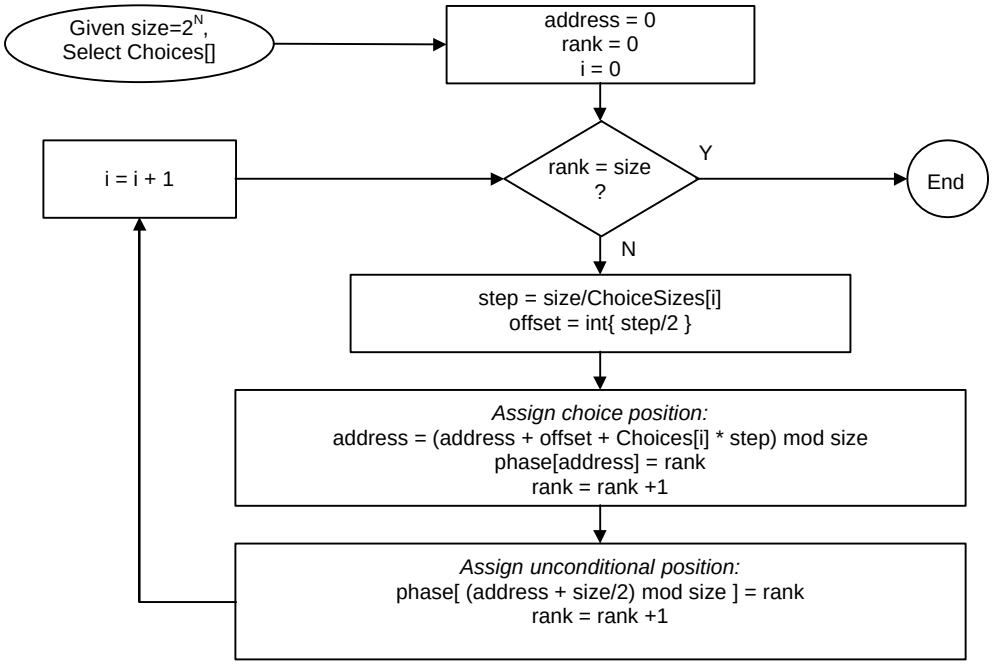

Figure 8. Phase Array Generator.

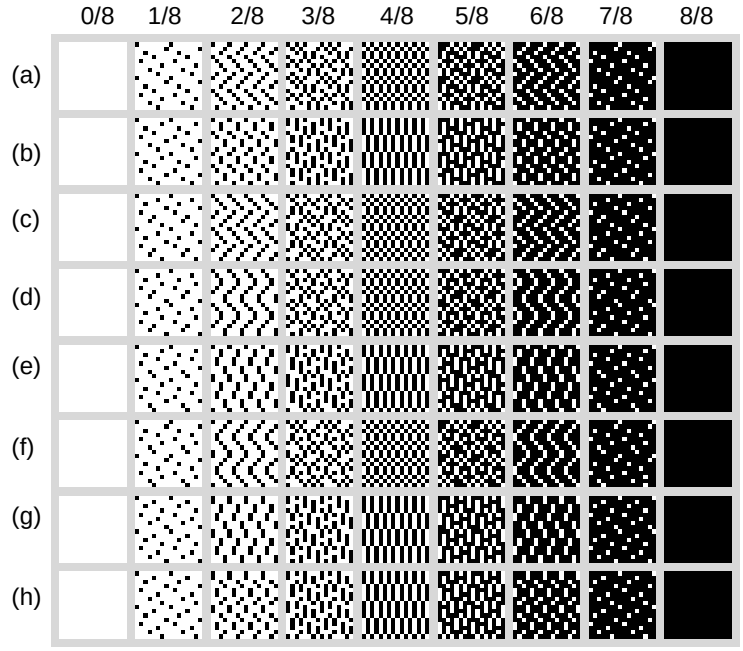

Figure 9. 8x8 element dither patterns for different phase arrays.

5. EXAMPLES

Table 2 indicates that there are 8 total unique phase arrays for an array size of 8. Although the choice size vector is [8 2 4 2], redundancy due to translation and reflection are eliminated by always forcing the first two choice values to 0. Using the algorithm of Figure 8, the 8 unique phase arrays are:

	Phase Array	Choice Vector
(a)	[0 7 2 4 1 6 3 5]	[0 0 0 0]
(b)	[0 6 2 4 1 7 3 5]	[0 0 0 1]
(c)	[0 5 2 7 1 4 3 6]	[0 0 1 0]
(d)	[0 5 2 6 1 4 3 7]	[0 0 1 1]
(e)	[0 6 2 5 1 7 3 4]	[0 0 2 0]
(f)	[0 7 2 5 1 6 3 4]	[0 0 2 1]
(g)	[0 4 2 6 1 5 3 7]	[0 0 3 0]
(h)	[0 4 2 7 1 5 3 6]	[0 0 3 1]

The dither patterns resulting from applying these arrays to the systems in Figure 5 are illustrated in Figure 9. As before, a 16x16 pixel image is shown for each of the 9 gray levels. The reason for this is that the dither patterns have a period of 8x8 pixels and showing 4 periods in this way allows full examination of the wrap-around properties of the patterns. It is particularly interesting to note that the phase array in phase array (g) is the same as the bit-reversed dither array, and Figure 9(g) is the same as the patterns in Figure 6(b). Also, the phase array build in the example of Figure 8, [0 7 2 5 1 6 3 4], is shown in Figure 9(f).

It is clear that the 8 phase arrays listed above are unique in that none of them are simple vertical translations or reflections of another. However upon examination of Figure 9, it can be observed that for every phase array, there is set of patterns with another phase array that differ only by a 2-D translation. In this 8x8 case of Figure 9, the following pairs of patterns are essentially identical: phases (a) and (c); phases (b) and (h); phases (d) and (f); and phases (e) and (g). This redundancy is a complex interaction with the horizontal dither array and predicting it based on the phase array values in not

obvious. In general, it is found that the redundancy factor for an array of size 2^N is 2^{N-2}. So, for size=8 there will be sets of 2 copies. For size=32 there will be sets of 8 copies.

This technique is particularly useful for arrays of size 32. Figure 10 illustrates the dither patterns (all black excluded) when the bit-reversed phase array, [0 16 8 24 4 20 12 28 2 18 10 26 6 22 14 30 1 17 9 25 5 21 13 29 3 19 11 27 7 23 15 31], was used .

All of the dither patterns form the 33,554,432 unique phase arrays were analyzed for homogeneity and the following one was found to give very good results: [0 22 12 30 4 19 10 24 2 21 15 28 6 16 9 27 1 23 13 31 4 18 11 25 3 20 14 29 7 17 8 26]. The patterns associated with this phase array are shown in Figure 11. Both Figure 10 and Figure 11 show 4 periods of each gray level (64x64 pixels) to examine the wrap-around properties of the patterns.

In our hardware implementation of Figure 1 the effective 32x32 dither system using the phase array of Figure 11 was used to dither 24-bit color video to 12-bits. After pass through the 1-D inverse dither system, the output is in most cases perceptually indistinguishable from the input.

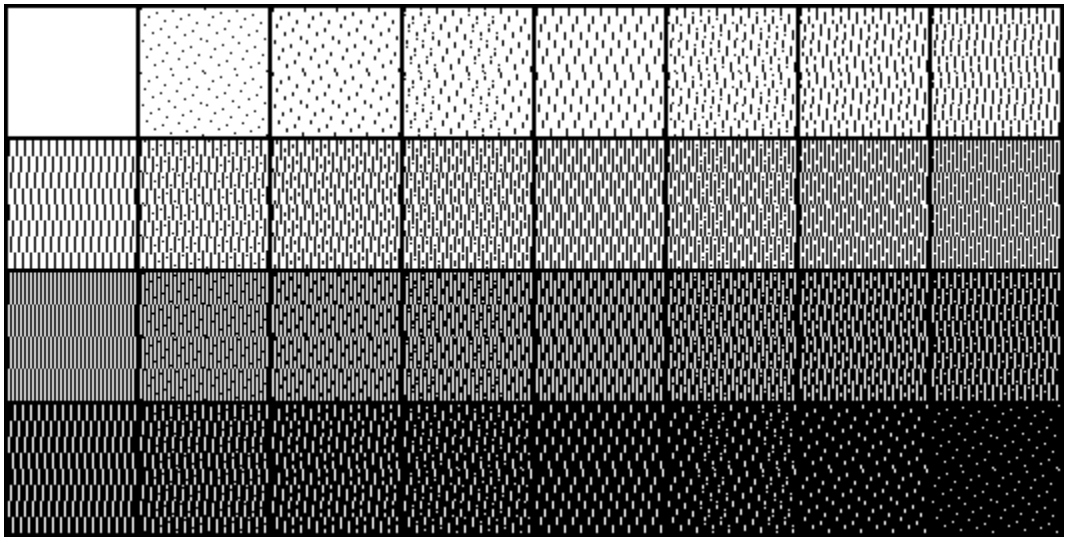

Figure 10. 32x32 element dither patterns using bit-reversed phase array, [0 16 8 24 4 20 12 28 2 18 10 26 6 22 14 30 1 17 9 25 5 21 13 29 3 19 11 27 7 23 15 31]

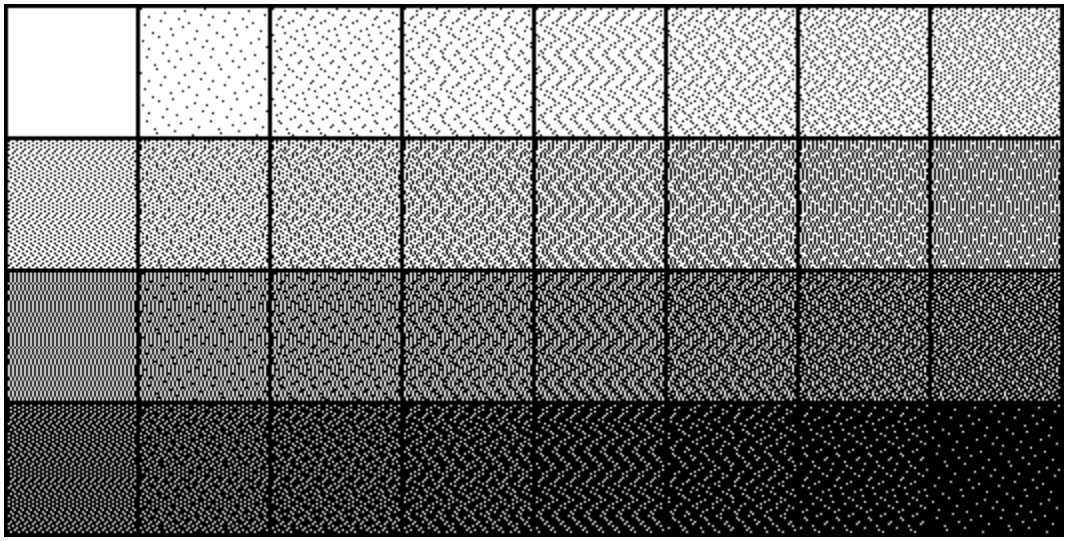

Figure 11. 32x32 element dither patterns using an optimum phase array, [0 22 12 30 4 19 10 24 2 21 15 28 6 16 9 27 1 23 13 31 4 18 11 25 3 20 14 29 7 17 8 26]

REFERENCES

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

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

[3] R. Ulichney, "Video Rendering", *Digital Technical Journal*, vol. 5, no. 2, pp. 9-18, 1993.

[4] Z. Xiong, M. T. Orchard and K. Ramchandran, "Wavelet-Based Approach to Inverse Halftoning," *Proc. of IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, 1997.

[5]] R. L. Stevenson, “Inverse Halftoning via MAP Estimation,” *IEEE Transactions on Image Processing*, vol. 5, no. 4, pp. 574—583, April 1997.

[6] A. C. Barkans, "Color Recovery: True-Color 8-Bit Interactive Graphics," *IEEE Computer Graphics and Applications*, vol. 17, no. 1, January/February 1997.

[7] C. M. Miceli and K. J. Parker, "Inverse Halftoning," *Journal of Electronic Imaging*, vol. 1, no. 2, pp. 143—151, 1992.

[8] R. Eschback and R. Hauck, "Analytic description of the 1-D error diffusion technique for halftoning", *Optic Comm*., vol. 52, no. 3, pp. 165-168, 1984.

[9] D. Neuhoff, T. Pappas, and N. Seshadri, "One-dimensional least-squares model-based halftoning", *Proc. ICASSP-92*, 1992.

[10] W. Press, B. Flannery, S. Teukolsky, W. Vetterling, *Numerical Recipes in C*, Cambridge University Press, section 12.2 "Fast Fourier Transform (FFT)", 1988.