7-1994

Energy-Based Evaluation of Digital Halftones

John Weible
Pittsburg State University

ENERGY-BASED EVALUATION OF DIGITAL HALFTONES

Submitted to the Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

by

John Weible

PITTSBURG STATE UNIVERSITY

Pittsburg, Kansas

July, 1994

Acknowledgments

I would like to thank two people in particular, for their help during the writing of this paper. I thank Dr. William Studyvin, my advisor, without whom I would probably have never started. I am also especially grateful for my wife Cherie', without whom I would probably never have finished.

ENERGY BASED EVALUATION OF DIGITAL HALFTONES

An Abstract of the Thesis by
John Weible

The purpose of this study was to determine the validity of the energy measure developed by Geist, Reynolds, and Suggs, when used as an evaluator of digitally halftoned images. The energy measure was found to be a valid, useful tool for the evaluation of binary digital halftone quality. Data resulting from the analysis and visual comparison of fifteen different halftones supports this conclusion. Using linear regression, the coefficient of correlation between the energy measure and visual quality ratings was -0.606 using all images, and -0.936 using average results for each halftone method. These figures indicate the strong relationship between image energy and image quality.

Although the energy measure was found to be accurate for different halftones of the same continuous-tone image, there is an inherent difficulty when comparing the quality of halftones of different image content. Geist, Reynolds, and Suggs' algorithm does not produce values within a fixed

range. A simple approximation for normalizing the energy values is proposed and used for the study, but further development is needed to obtain absolute quality rankings using this technique.

TABLE OF CONTENTS

CHAPTER PAGE

I. INTRODUCTION 1
The Origins of Halftoning 1
Types of Digital Halftoning 2
Comparing Halftones 4
Statement of the Problem 5
Objectives 6
Significance of the Study 6

II. DEVELOPMENTS IN DIGITAL HALFTONING 8
Standard for Comparison 8
Thresholding 10
Ordered Dither 12
Error Diffusion 16
Dot Diffusion 23
White Noise 24
Blue Noise 26

III. HALFTONING 31
Evaluating Halftones 31
Design of the Study 34
Methodology 36

IV. RESULTS AND ANALYSIS 38
Obtaining the Data 38
Results 38
Interpretation of the Data 41

V. SUMMARY AND CONCLUSIONS 45
Summary of Results 45
Conclusions from the Study 46

VI. RECOMMENDATIONS 48

BIBLIOGRAPHY 50

APPENDIX A. Related Procedures in Image Processing 55

APPENDIX B. Evaluation Criteria and Data Sheet 58

APPENDIX C. Test Images 61

APPENDIX D. Software Source Code 83

APPENDIX E. The Energy Measure 103

APPENDIX F. Error Diffusion Filters 105

LIST OF TABLES

Table I. Visual Evaluation Results 40

Table II. Energy Measure Results 43

Table III. Normalized Energy Measure Results 43

Table IV. Evaluation Summary by Image 44

Table V. Averages of Results by Halftone Method 44

Table VI. Error Diffusion Filters 106

LIST OF FIGURES

Figure 1. Sample Image for Comparison (Printed at high resolution of 300 dpi) 9

Figure 2. Results of Simple Thresholding (75 dpi) 11

Figure 3. Clustered-dot Ordered Dither (75 dpi) 13

Figure 4. Bayer Dispersed-dot Ordered Dither (75 dpi) 15

Figure 5. Floyd-Steinberg Error Diffusion (75 dpi) 19

Figure 6. Burkes Error Diffusion (75 dpi) 20

Figure 7. Jarvis, Judice, & Ninke Error Diffusion (75 dpi) 21

Figure 8. Stucki Error Diffusion (75 dpi) 22

Figure 9. White Noise Random Dither (75 dpi) 25

Figure 10. Floyd-Steinberg Error Diffusion, with Serpentine Raster (75 dpi) 28

Figure 11. Floyd-Steinberg Error Diffusion, with 50% Random Weights and Serpentine Raster. (75 dpi) 29

Figure 12. Image #1. Prescaled Halftone for Reference. (300 dpi) 63

Figure 13. Image #1. Bayer Ordered Dither. (75 dpi) 64

Figure 14. Image #1. Jarvis, Judice, & Ninke Filter. (75 dpi) 65

Figure 15. Image #1. 50-Percent Random Floyd-Steinberg Filter. (75 dpi) 66

Figure 16. Image #2. Prescaled Halftone for Reference. (300 dpi) 67

Figure 17. Image #2. Bayer Ordered Dither. (75 dpi) 68

Figure 18. Image #2. Jarvis, Judice, & Ninke Filter. (75 dpi) 69

Figure 19. Image #2. 50-Percent Random Floyd-Steinberg Filter. (75 dpi) 70

Figure 20. Image #3. Prescaled Halftone for Reference. (300 dpi) 71

Figure 21. Image #3. Bayer Ordered Dither. (75 dpi) 72

Figure 22. Image #3. Jarvis, Judice, & Ninke Filter. (75 dpi) 73

Figure 23. Image #3. 50-Percent Random Floyd-Steinberg Filter. (75 dpi) 74

Figure 24. Image #4. Prescaled Halftone for Reference. (300 dpi) 75

Figure 25. Image #4. Bayer Ordered Dither. (75 dpi) 76

Figure 26. Image #4. Jarvis, Judice, & Ninke Filter. (75 dpi) 77

Figure 27. Image #4. 50-Percent Random Floyd-Steinberg Filter. (75 dpi) 78

Figure 28. Image #5. Prescaled Halftone for Reference. (300 dpi) 79

Figure 29. Image #5. Bayer Ordered Dither. (75 dpi) 80

Figure 30. Image #5. Jarvis, Judice, & Ninke Filter. (75 dpi) 81

Figure 31. Image #5. 50-Percent Random Floyd-Steinberg Filter. (75 dpi) 82

CHAPTER I

INTRODUCTION

The Origins of Halftoning

Halftoning, regardless of the medium, refers to methods of displaying a reasonable reproduction of an image, while using fewer component colors than the original (Gentile, Walowit, & Allebach, 1990, p. 1019). In many cases, the number of available colors with which we intend to create a halftone is very limited -- often only black and white (Mitsa & Parker, 1992, p. 1920).

Methods of "halftoning" have existed for centuries, although the term was coined much later, to describe a particular printing process. In 1880, Stephen Hargon invented the printing of "halftones" by photoengraving, a process similar to modern screen printing techniques (Rogers, 1985, p. 102).

Long before modern printing was developed, halftone-like methods could be found in several types of artwork. Woodcuts, etchings, and pen-and-ink drawings often consist of black ink on white paper. Many patterns of lines, contours, and hatch-marks are used to give the impression of

shades. All of these techniques have the same purpose as what is specifically called halftoning.

The fundamental capabilities and limitations of halftoning were described by Gentile, Walowit, and Allebach (1990, p. 1020) as relying

> "...on the viewer's making a local spatial average over patterns of alternating colors to create the impression of a color that lies between those that compose the pattern. The resulting increase in the number of perceived colors that can be displayed or printed is achieved at the expense of decreased spatial resolution and, in some cases, at the expense of the appearance of artifacts in the image."

Types of Digital Halftoning

Digital halftoning techniques fall into several categories. With any particular technique, the pixels (individual image dots) are calculated based on either points or neighborhoods. Each particular method will also produce either clustered-dot or dispersed-dot patterns, and either periodic or aperiodic patterns (Ulichney, 1987, p. 3).

Point vs. Neighborhood Algorithms

If the value of each pixel in the final halftone depends only on its position and that single pixel's original intensity, a "point" algorithm is being used. On the other hand, a "neighborhood" algorithm also considers the intensities of nearby pixels when calculating each point. Therefore, neighborhood methods are more computationally complex, but achieve better results (Ulichney, 1987).

Clustered-dot vs. Dispersed-dot Algorithms

Clustered-dot algorithms simulate the traditional printing halftone screen, by grouping adjacent pixels together to simulate varying sizes of dots. Dispersed-dot methods attempt to isolate the pixels within the halftone patterns, to create smoother shading overall. Dispersed-dot halftones also have greater effective resolution, but are not effective if the output device does not adequately accommodate isolated pixels (Peli, 1991, p. 625; Ulichney, 1987; Rogers, 1985).

Periodic vs. Aperiodic Algorithms

Periodic halftone algorithms work by overlaying a repeated array of numbers, called a mask, on the image to

introduce a "dithering" noise. These periodic algorithms are generally faster than aperiodic algorithms, and they lend themselves to parallel implementations. A major drawback is that the periodic process itself usually produces unwanted side effects in the image (Ulichney, 1987; Geist, Reynolds, & Suggs, 1993; Peli, 1991, p. 625).

Comparing Halftones

The selection of a halftoning method in a given situation is often a combination of skill and trial-and-error. Some techniques are acceptable with certain images and not others. The resolutions of the image itself and of the output device also has considerable bearing on the results. For example, clustered-dot algorithms inherently reduce the apparent resolution in the image. Small details can be completely lost. If, however, the output device has very high resolution, excellent clustered-dot halftones can be produced (Peli, 1991, p. 625; Linotype-Hell Co., 1993). This is why photographs or artwork reproduced by screen printing may not appear halftoned at all.

In many cases, the equipment available is not capable of very high resolution. Then the particular method of halftoning used may drastically alter the appearance and

quality of an image. Until quite recently, visual inspection and expertise with halftoning methods were the only ways to determine the success of a particular halftone (Ulichney, 1987; Geist, Reynolds, & Suggs, 1993, p. 137; Peli, 1991, p. 625).

The quality of halftones can be measured a number of ways, but there are difficulties with nearly every one of them. Qualitative comparisons by eye are naturally subject to the viewer's bias. Quantitative methods have recently become prevalent in the analysis of halftones. Some of these quantitative methods, however, do not produce definitive results. A definitive, numerical method for evaluating the quality of halftones is desired (Geist, Reynolds, & Suggs, 1993; Ulichney, 1987).

Statement of the Problem

The goal of this research is to determine the validity of Geist, Reynolds, and Suggs' energy measure as a tool for evaluating binary halftones. This will be tested by finding the correlation between visual quality rankings and the energy for a variety of standard digital halftones.

Objectives

1. Qualitatively compare five images produced by each of three methods:

   A) Bayer ordered dither

   B) Jarvis, Judice, and Ninke's minimal average error

   C) Ulichney's 50%-random-weighted error diffusion

   The accuracy of image details, edges, contrast, intensity, and the presence of any artificial patterns or other distortions in each image will be rated numerically on a checklist (See Appendix B for an example checklist and the specific criteria).

2. Evaluate the same halftone images quantitatively using the energy measure algorithm of Geist, Reynolds, and Suggs.

3. Correlate the averaged qualitative results with the energy calculations to determine the validity of the energy measure.

Significance of the Study

Geist, Reynolds, and Suggs (1993, pp. 153-154) did not demonstrate the accuracy of their energy measure for comparison of advanced digital halftoning algorithms, using a variety of images. Energy data were only reported for the

halftones of a single image. The energy measure was introduced in "A Markovian Framework for Digital Halftoning," but it was not the primary focus of the paper. Attention was instead focused on two halftoning methods and their fundamental equivalence.

If shown to be accurate, the energy measure's significance as a comparison tool will lie in its ease of use: it yields a single deterministic number for any given halftone image. The power spectra used by Ulichney, for example, are much more difficult to evaluate.

CHAPTER II

DEVELOPMENTS IN DIGITAL HALFTONING

Standard for Comparison

To illustrate the different techniques of halftoning as they are discussed, sample halftones are included. Except for Figure 1, all the following images are printed at a relatively low resolution of 75 dots per inch (dpi), which allows the viewer to discern individual pixels in the halftones, if desired. The same input file was processed by each of the halftoning methods.

A close approximation of the original sample image is shown in Figure 1 for reference. The image began as a full-color scan of a photograph. It was then converted to 256 levels of gray. To print the image here, it was halftoned using a clustered-dot technique, which simulates a traditional 45° halftone screen with 60 "intensities." Figure 1 is a fairly accurate representation of the original, since its resolution was scaled up by a factor of four before computing the halftone (See the discussion of prescaling in Appendix A, page 56).

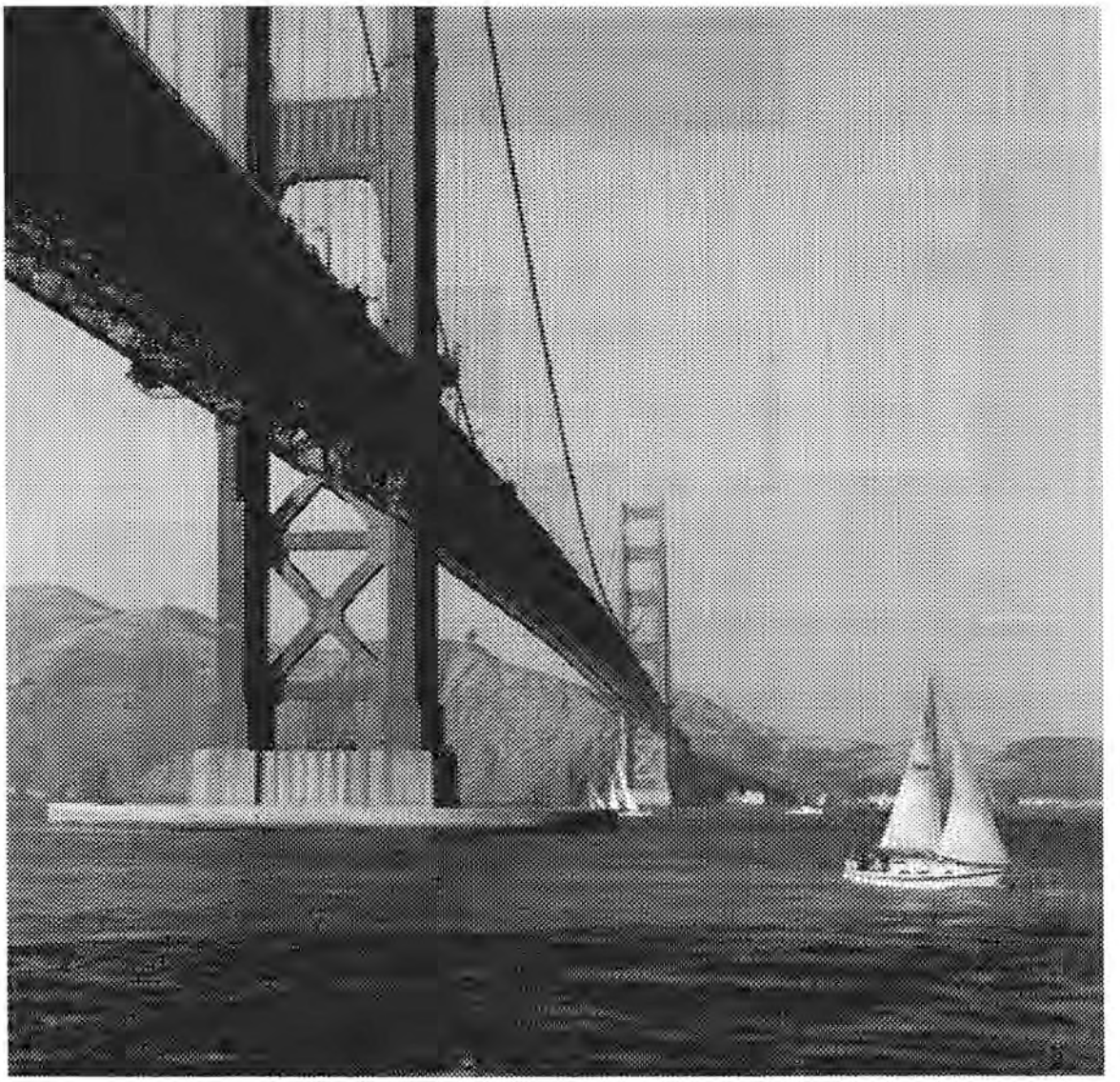

Figure 1. Sample Image for Comparison (Printed at high resolution of 300 dpi)

Thresholding

Thresholding is the simplest method of halftoning used. In a sense, it is not even a method of halftoning, but rather the result of not halftoning. It entails simply replacing each pixel in the original with the nearest color available for the halftone. This technique is also occasionally called "rounding," as the pixel value is rounded to the nearest possible value. Thresholding is a local technique, since the value of each halftone pixel is unaffected by any others. Thresholding yields unacceptable results for most applications, since nearly all information in the original image is lost. (Geist, Reynolds, & Suggs, 1993) Figure 2 illustrates the poor results of simple thresholding.

Thresholding is often used when producing halftones containing more than two colors, however. The more colors or intensities available for the reproduction of an image, the less need there is for halftoning. This perhaps explains why the majority of halftoning research has focused on either bilevel (black-and-white) or four-color (usually cyan, magenta, yellow, and black) halftones.

Figure 2. Results of Simple Thresholding (75 dpi)

Ordered Dither

Dither refers to the introduction of noise to an image. In the case of ordered dither, the noise is repetitive and well defined, thus "ordered." This contrasts with both white noise random dither and random-weighted error diffusion, which are unordered dithering techniques.

Clustered-dot Ordered Dither

The clustered-dot ordered dither essentially simulates a classical optical screen. It uses a matrix of pixels to create the varying sizes of dots characteristic to optical screens. The number of sizes of dots available, and hence the number of apparent intensities, is dependent on the number of pixels in each matrix. Figure 1 for example, was dithered with an 8x8 matrix yielding 60 different "intensities." Figure 3 is also a clustered-dot ordered dither, but using the same 75 dpi as the other images to allow a fair comparison.

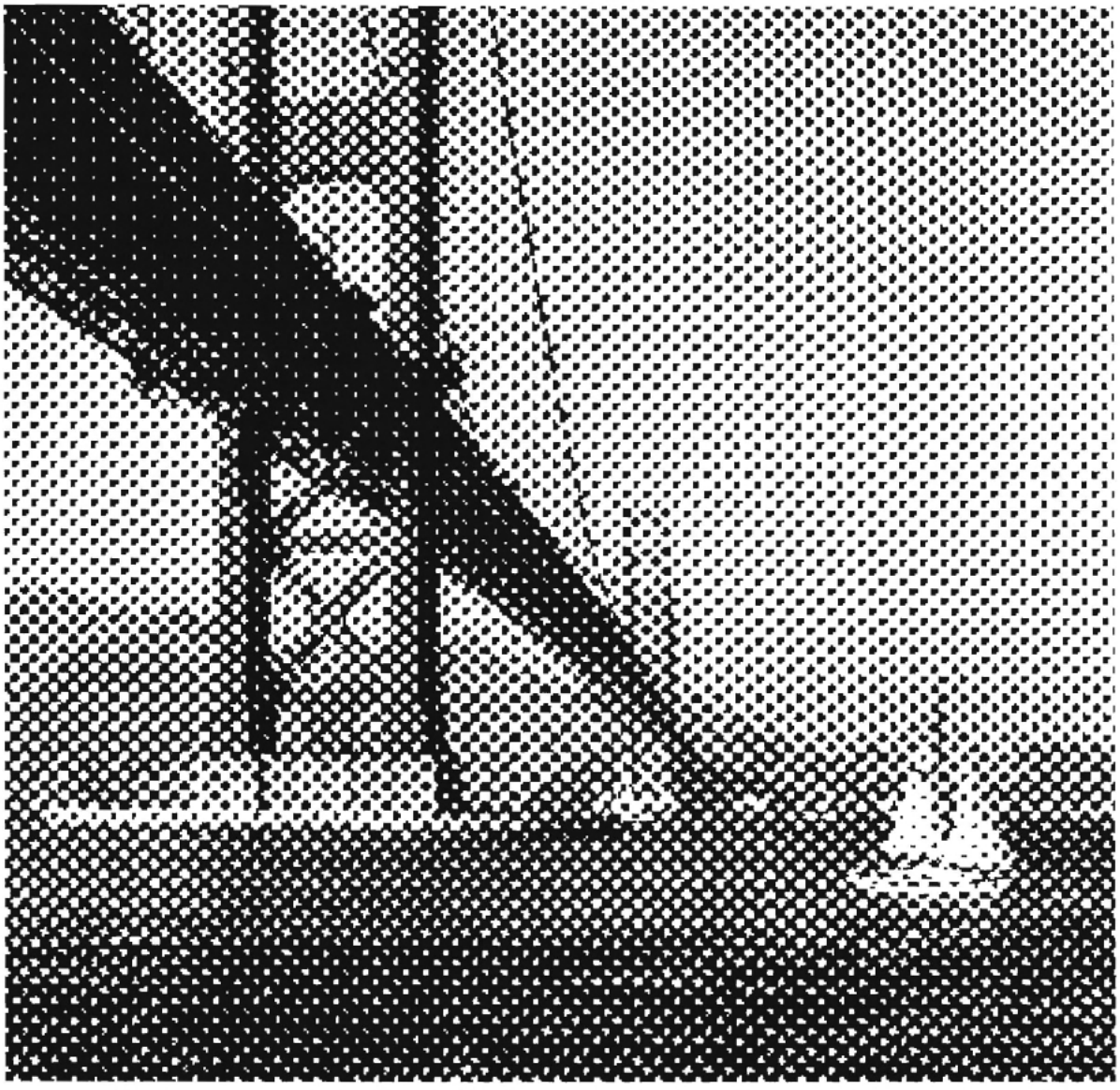

Figure 3. Clustered-dot Ordered Dither (75 dpi)

As is apparent from this image, most details in the image were lost in the halftoning process. Even some details that were apparent in Figure 2 have been lost. Note in particular the vagueness of the hills in the background and the distant bridge upright. The vertical support cables are not visible at all. Because this method requires so many

pixels to simulate each halftone "dot," it generally produces good results only with high resolution devices (Rogers, 1985, pp. 102-104; Peli, 1991, p. 625). One benefit of the technique is that the simulated "screen" is so consistent across the entire image, that it does not detract visually as much as some other halftoning artifacts. The familiarity of viewers to the technique also increases its acceptance (Ulichney, 1987).

Bayer Dispersed-dot Ordered Dither

In 1973, Bayer demonstrated a more successful type of ordered dither. Instead of "clustering" the dots to simulate an optical screen, the dots were "dispersed" to smooth out the image.

In the Bayer dither, a standard, predetermined matrix of thresholds is used. The matrix is superimposed in a repeating grid on the image. For each pixel, the halftone value is determined simply by comparing its original intensity to the corresponding threshold in the matrix. The primary advantage of ordered dither over clustered-dot techniques is that the effective resolution is not reduced when the matrix size is increased (Bayer, 1973; Rogers, 1985).

Figure 4 was halftoned with an 8x8 dither matrix, which allows the simulation of 64 intensities. The Bayer algorithm is still commonly used since it is fast, simple, and may be calculated in parallel.

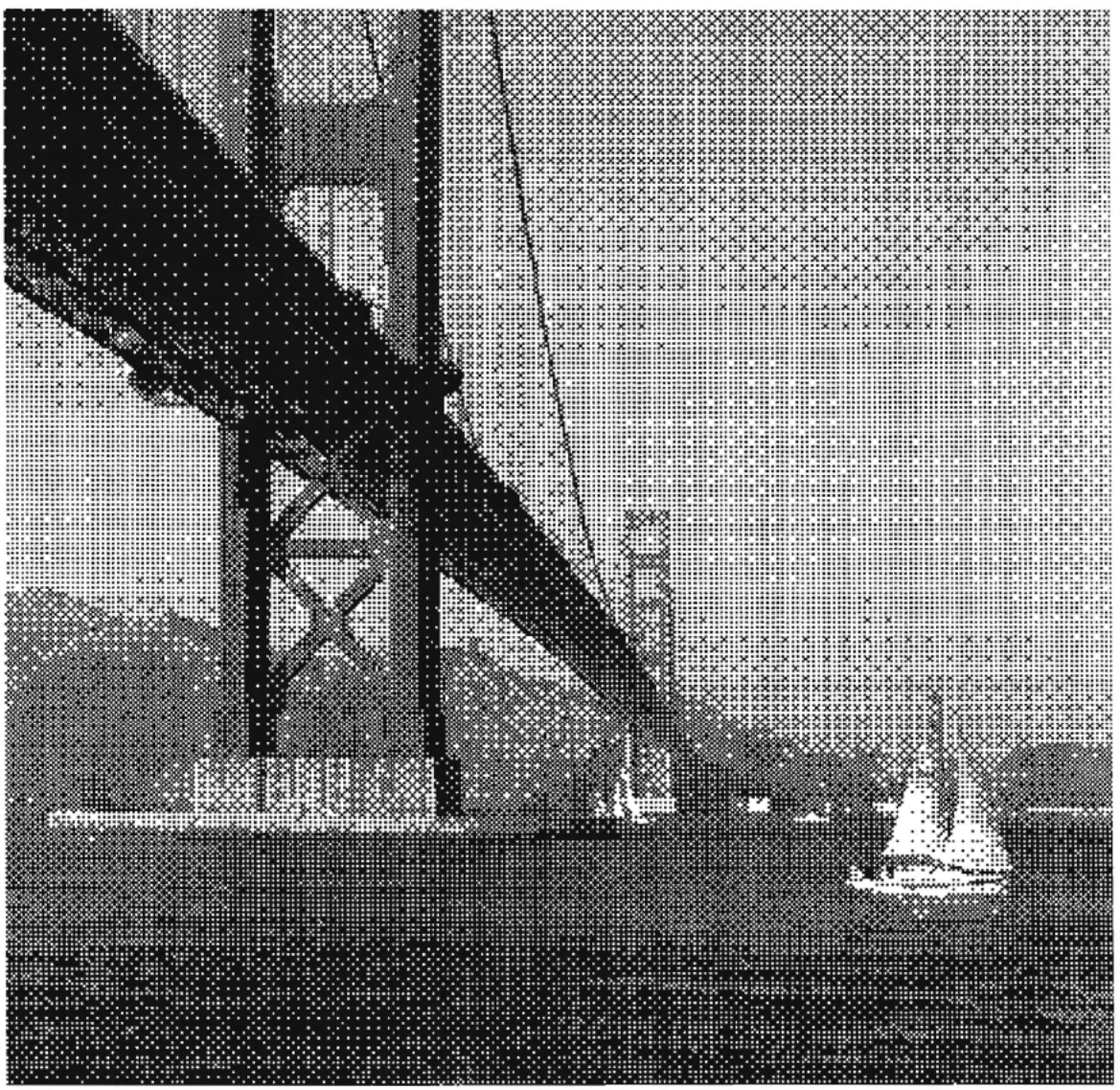

Figure 4. Bayer Dispersed-dot Ordered Dither (75 dpi)

Error Diffusion

Whenever halftoning is necessary, the final image must differ, at least slightly, from the original. At any individual pixel, however, the halftone may or may not differ from the original. The actual difference is referred to as the "error." Naturally, the visual error should be kept small in the halftoning process. Techniques previous to error diffusion, including thresholding and ordered dithers, do not attempt to minimize the overall error.

Floyd and Steinberg introduced an elegant method of halftoning, based on the idea of distributing or "diffusing" the error, which results from thresholding each pixel, to its neighbors. They called it "error diffusion" (Floyd & Steinberg, 1975).

Error diffusion is the first "neighborhood" process introduced. For most purposes, it produces superior halftones to previous techniques, without their characteristic periodic patterns.

There are a few drawbacks, however. Error diffusion is slower than the techniques mentioned previously. Different types of unwanted patterns may develop in regions of similar intensities and near boundaries (Ulichney 1987, pp. 242-

253). In the following example images, these artifacts are most noticeable in the sky. The severity of these artifacts depends on the image and the error diffusion filter employed.

Another problem with this method of error diffusion is due to the raster processing order of the algorithm. Low-contrast edges and gradations are distorted toward the lower right corner of the image. Ulichney has called this "directional hysteresis." (Ulichney 1987, pp. 242-253)

The error diffusion technique considers each pixel in the image moving from top to bottom, processing each line left to right. Each pixel is compared to a threshold. The "error," or difference between the desired intensity and the halftone intensity, is distributed in a weighted fashion to four adjacent pixels. See Figure 5 for a representation of the Floyd-Steinberg filter (a matrix of weights) and its results.

The dot represents the pixel being halftoned at a particular instant, and the numbers indicate the relative amount of error to distribute to each pixel. Since the four numbers sum to sixteen, each weight must be divided by sixteen before multiplying by the error. For example, the pixel immediately to the right will always receive $7/16^{ths}$ of

the error (Floyd & Steinberg, 1975; Ulichney, 1987, p. 239-241).

A number of researchers have attempted to improve on the original Floyd-Steinberg filter with different filters. Several notable examples follow. Each figure contains the example halftone for comparison, and is accompanied by the matrix of weights below. See Appendix F for further discussion of these filters.

Most of the error diffusion filters in the literature are significantly larger than the Floyd-Steinberg four-element filter. Jarvis, Judice, and Ninke (1976), Stevenson and Arce (1985), and Stucki (1979) all proposed 12-element filters (Rimmer, 1993, pp. 336-337). The larger filters tend to sharpen the images more, and increase the directional distortion (Ulichney 1987, p. 253). Although Jarvis, Judice, and Ninke termed their filter "minimal average error," it is included here, since the process is identical (Jarvis, Judice, & Ninke, 1976, p. 37; Ulichney, 1987, p. 253).

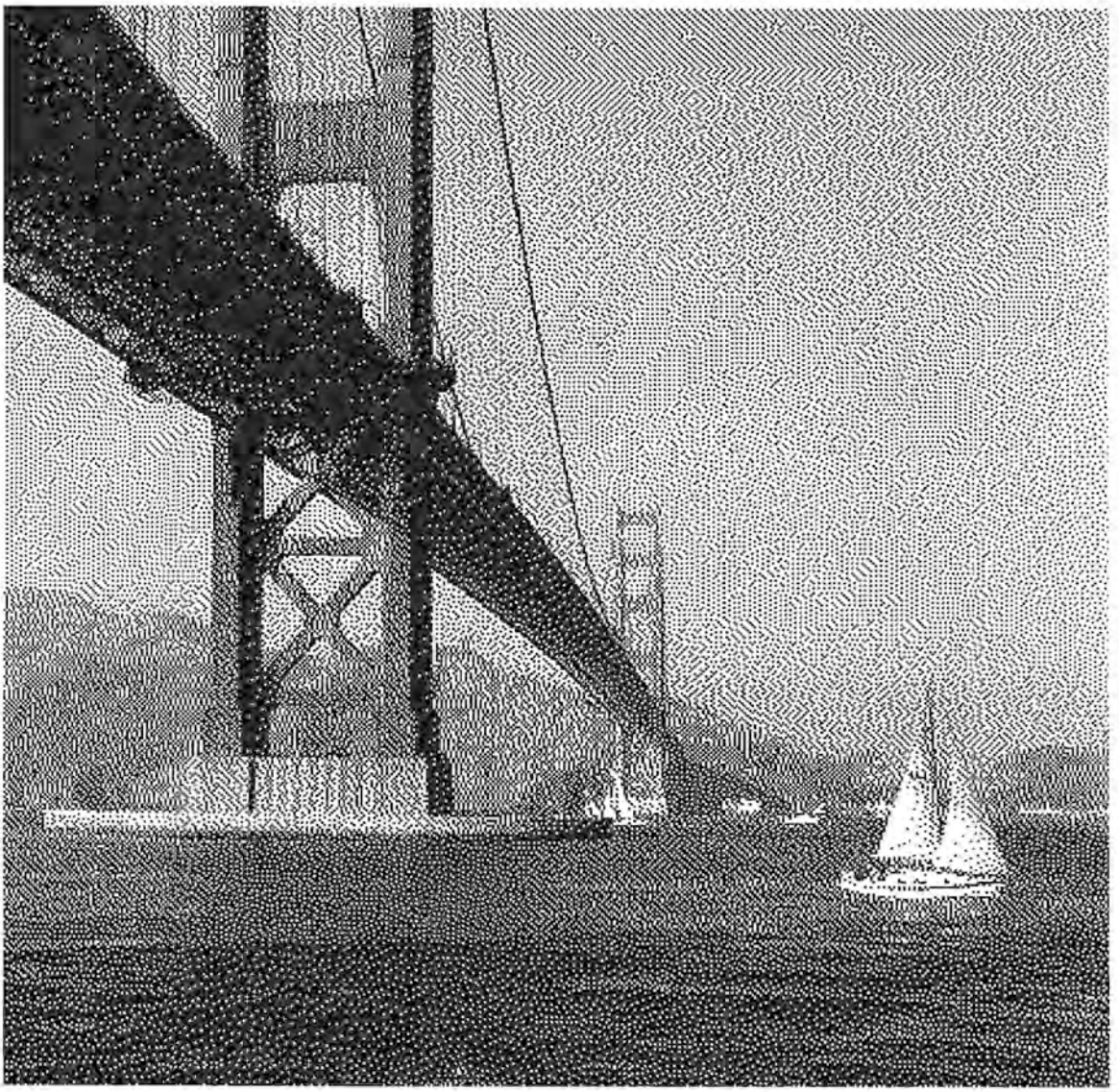

Figure 5. Floyd-Steinberg Error Diffusion (75 dpi)

• 7

3 5 1

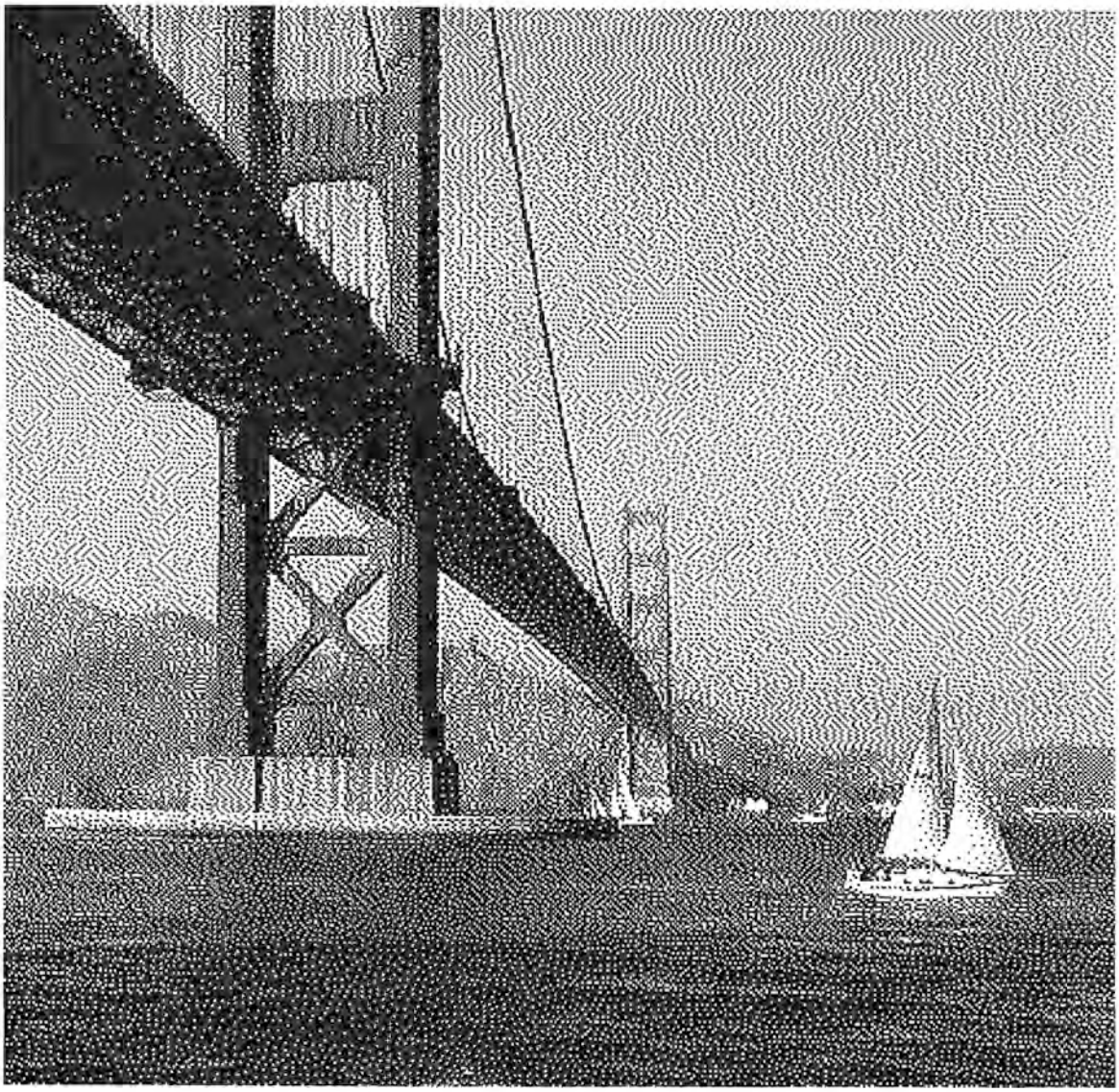

Figure 6. Burkes Error Diffusion (75 dpi)

• 8 4

2 4 8 4 2

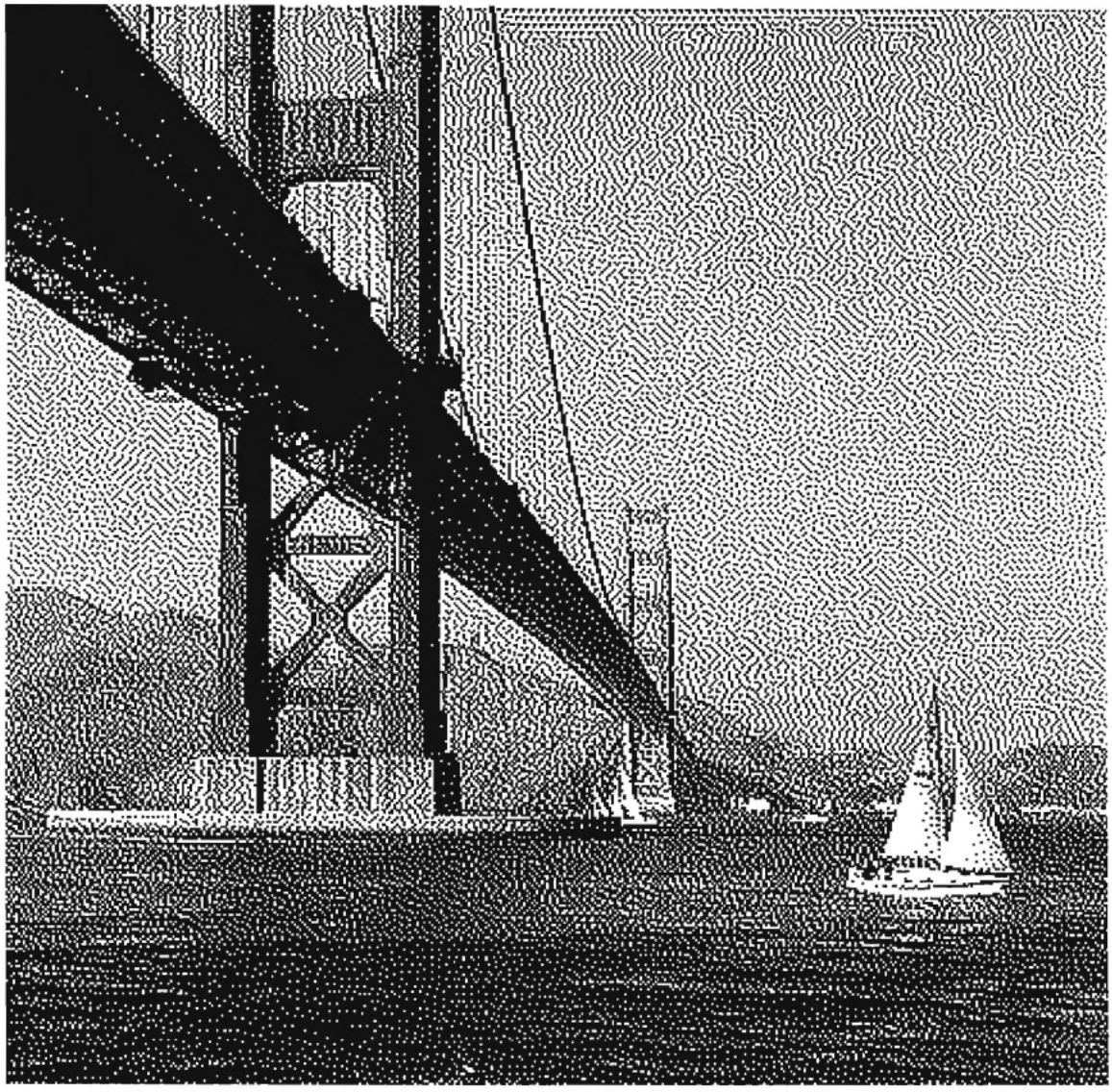

Figure 7. Jarvis, Judice, & Ninke Error Diffusion (75 dpi)

```
        • 7 5
    3 5 7 5 3
    1 3 5 3 1
```

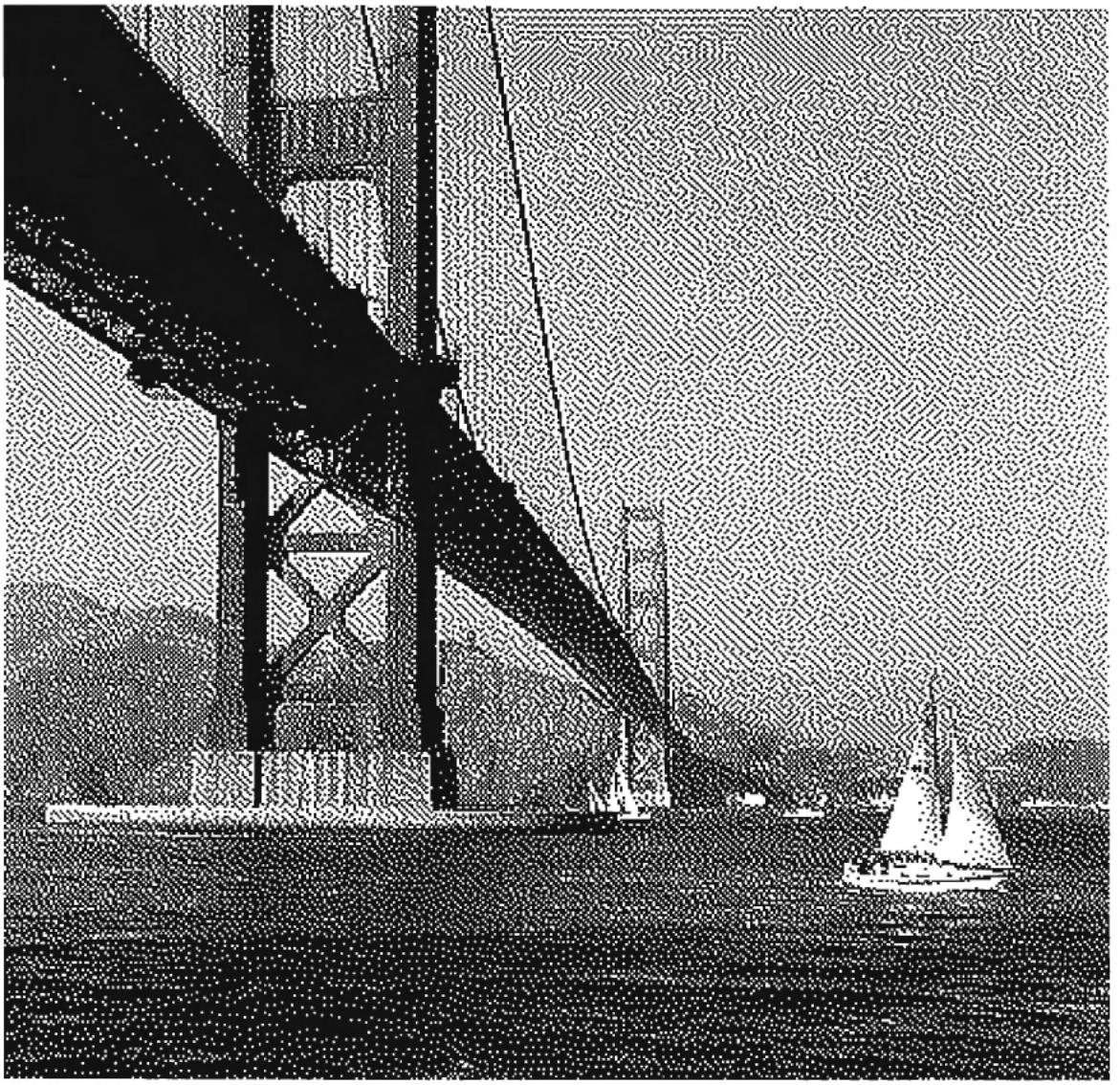

Figure 8. Stucki Error Diffusion (75 dpi)

```
        • 8 4
    2 4 8 4 2
    1 2 4 2 1
```

Dot Diffusion

Dot diffusion was developed by Donald Knuth as a way of employing the capabilities of error diffusion, while retaining the speed and parallel design inherent in the ordered dither algorithm (Geist, Reynolds, & Suggs, 1993, p. 139-140).

Dot diffusion superimposes a repeating matrix on the image to be halftoned, similar to the ordered dither. The values in the matrix represent diffusion weights, rather than dithered thresholds, however. The pixels are halftoned in the order of their corresponding weight, instead of scan line order. Error at each pixel is computed as in error diffusion, but the error is distributed to adjacent pixels with higher matrix weights (Geist, Reynolds, & Suggs, 1993, p. 139-140; Knuth, 1987).

Despite the diffusion used in the algorithm, the repeated matrix creates distracting patterns similar to the ordered dither. The dot diffusion algorithm is slightly more complex than error diffusion, but can take advantage of a multiprocessor system.

White Noise

White noise, by analogy to white light, is defined as having a spectrum that is approximately flat across the entire frequency range. White noise is completely random. To halftone using white noise means to use the standard thresholding technique, but make the threshold at each pixel a random number within the intensity range (Ulichney, 1987, p. 63).

White noise halftoning is also called "random dither" or occasionally "mezzotint," due to its resemblance to the seventeenth-century print making technique. As is apparent from Figure 9, white noise dither is not a viable halftoning method. It does, however, give a good basis for comparison to "blue noise" techniques (Ulichney, 1987, pp. 63-71).

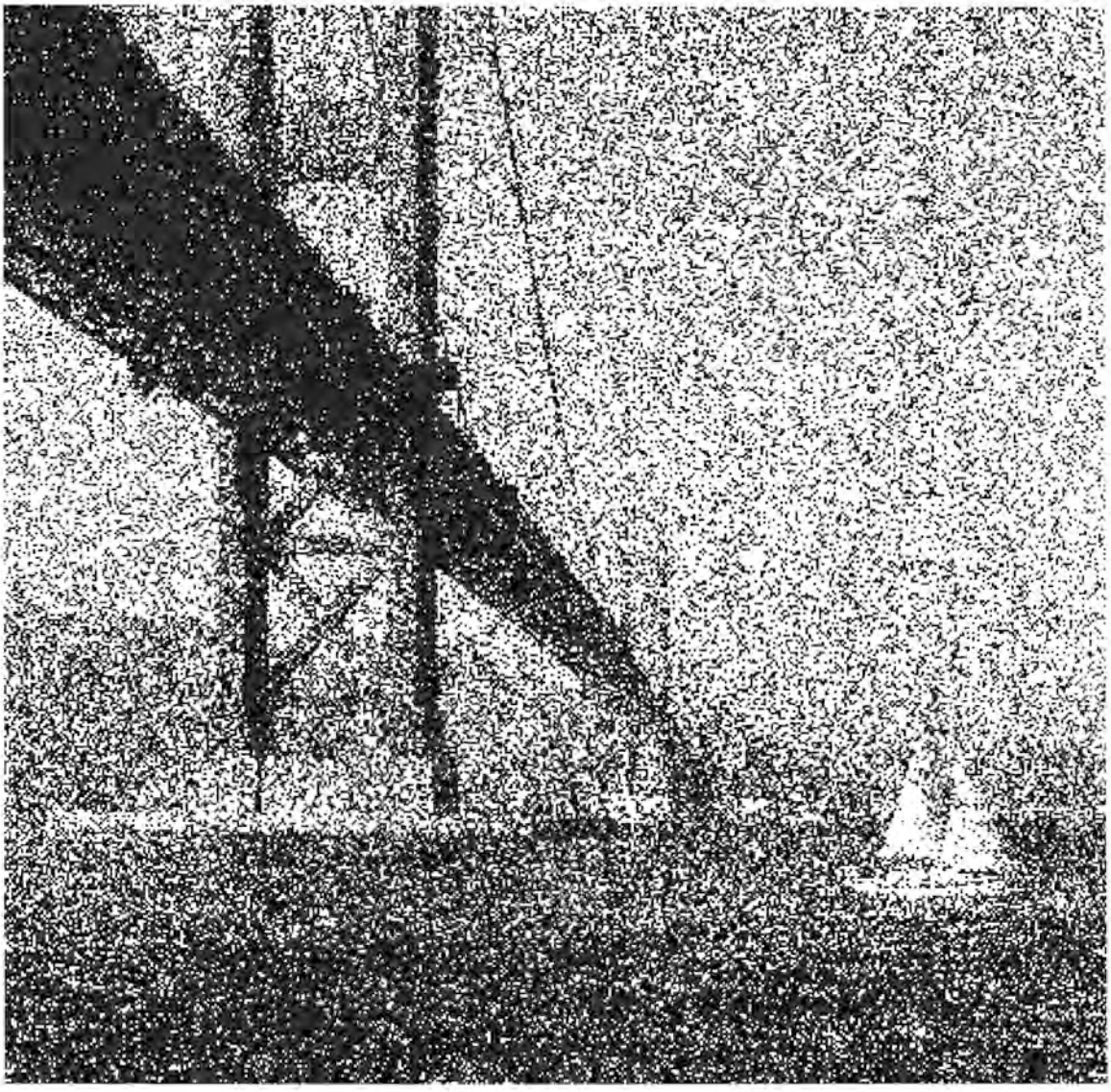

Figure 9. White Noise Random Dither (75 dpi)

Blue Noise

Ulichney's investigation of digital halftones showed that good halftone images have similar spectra. Their spectra have virtually no low-frequency components, and are nearly flat in upper frequencies. This type of frequency distribution is called "blue noise."

> Ulichney explains that, "being devoid of low frequencies and localized concentrations of spikes in the frequency domain, it [blue noise] has no structure and thus does not interfere with the interesting features of that which it is representing" (1987, p. 340).

The following halftoning algorithms were specifically designed to create images with good "blue noise" characteristics.

Improved Error Diffusion

Ulichney improved the results of the Floyd-Steinberg algorithm in two ways. First he suggested processing the pixels in a "serpentine" fashion, left-to-right, then right-to-left, alternating each scan line. This technique tends to produce different artifacts, but doesn't necessarily remove them, as can be seen in Figure 10 (Rimmer, 1993, p. 339).

Next, Ulichney added randomized weights to the error diffusion algorithm. The random weighting greatly reduces the artifacts (undesired patterns) often produced by standard error diffusion (Ulichney, 1987). This improvement is quite noticeable in the sky regions of Figure 11, especially near the top of the image.

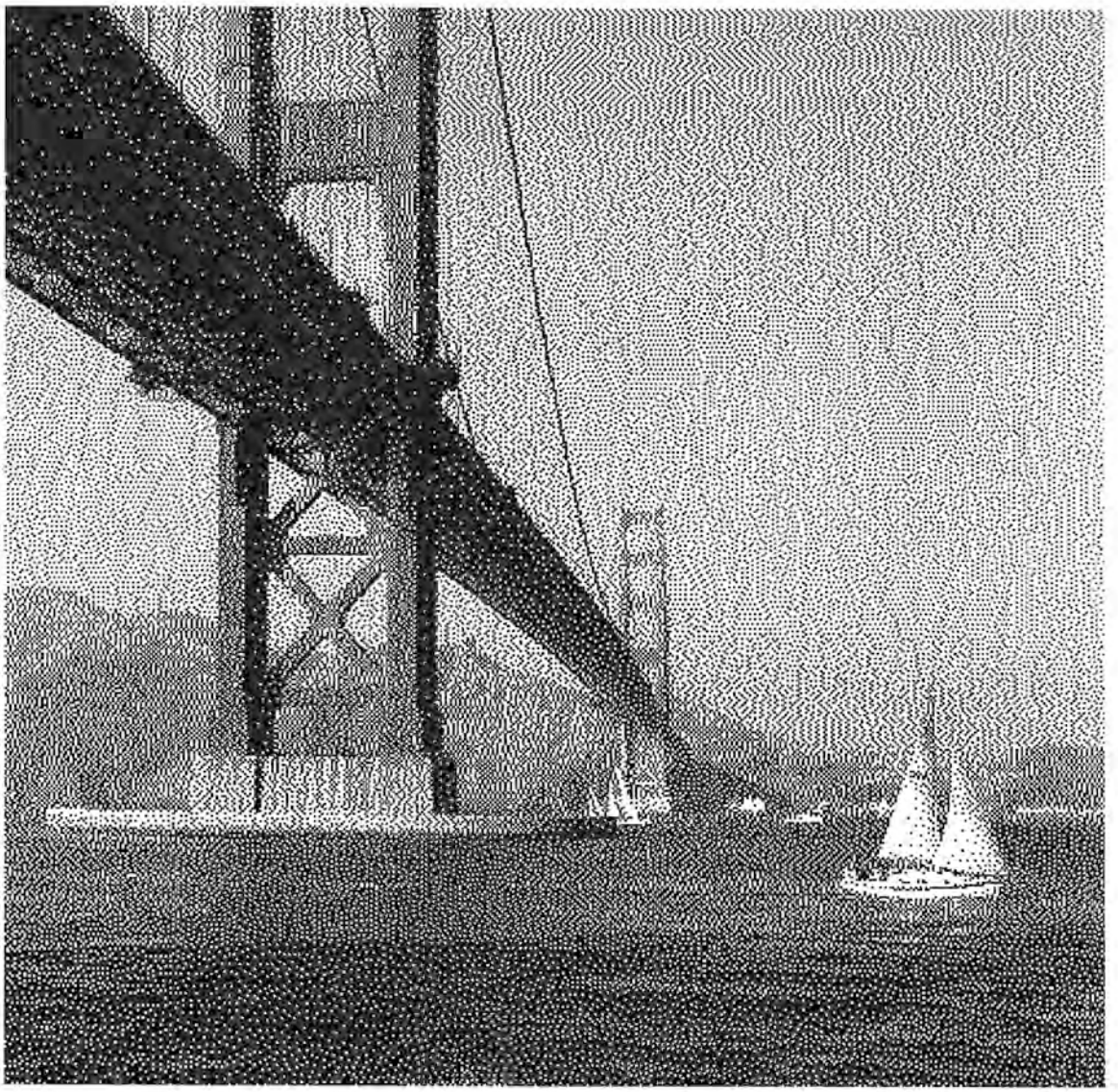

Figure 10. Floyd-Steinberg Error Diffusion, with Serpentine Raster (75 dpi)

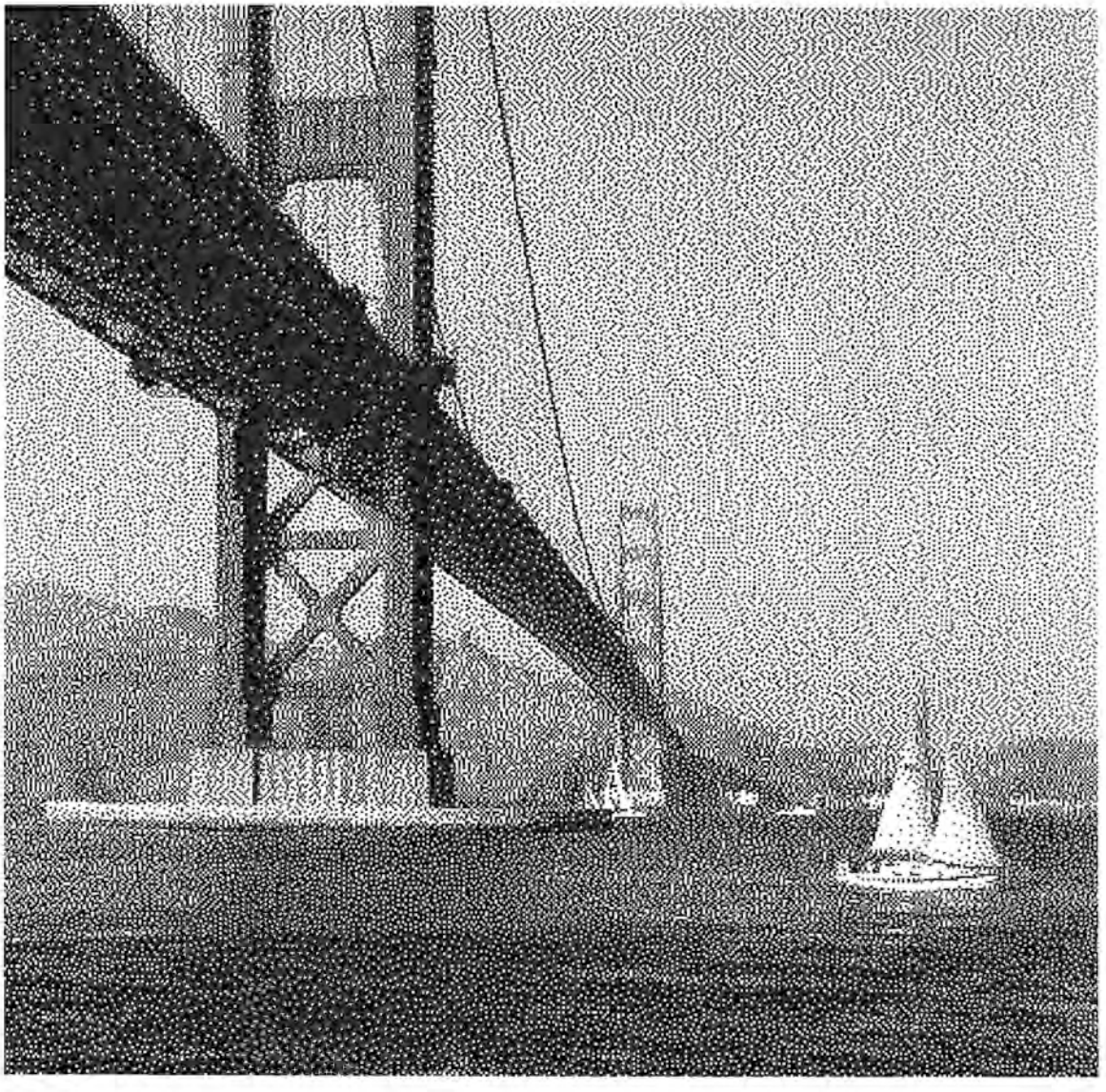

Figure 11. Floyd-Steinberg Error Diffusion, with 50% Random Weights and Serpentine Raster. (75 dpi)

Spectral-Based Algorithms

Since Ulichney's landmark book, Digital Halftoning, a number of halftoning methods have been developed, based on his spectral analysis. Mitsa and Parker (1992) achieved good results by creating "blue noise masks" and then using them similarly to a huge ordered dither matrix. Scheermesser, Broja, and Bryngdahl (1993) demonstrated an algorithm allowing adaptive spectral control over the halftoning process. Geist, Reynolds, and Suggs (1993) demonstrated excellent results with two equivalent algorithms, one using a neural network and one using simulated annealing.

These methods essentially work by inverting the power spectrum calculations Ulichney developed, in order to seek a halftone image that will satisfy the desired blue noise characteristics.

CHAPTER III

HALFTONING

Evaluating Halftones

Halftones are often evaluated visually by comparing any of several characteristics. We shall consider the original, unhalftoned digital image to be optimal. Most halftones give the viewer an approximate idea of the original image, but the success of any halftone is dependent on its faithfulness in representing the original, optimal image.

Any particular halftoning method may perform well on one image, but perform poorly on another. The evaluation and comparison of halftone methods, therefore, must involve a variety of test images.

Visual Comparison

A person can compare a number of characteristics of images by inspection. With similar pictures side-by-side, comparisons of contrast, detail, smoothness, and intensity can be made between them. Jarvis, Judice and Ninke noted that one advantage of the minimized average error method, (and therefore, error diffusion) over ordered dither is the

ability to enhance edges (1976, p. 13). Geist, Reynolds, and Suggs defined their subjective measure of quality as "a combination of sharpness of image detail and smoothness of gray-scale simulation." (1993, p. 148)

Additionally, unwanted textures or patterns may be apparent in an image. Shapes may be distorted, especially by error diffusion methods (Ulichney, 1987; Geist, Reynolds, & Suggs, 1993, pp. 138-141).

When considering halftoned images, the viewer should evaluate image quality by noting the characteristics mentioned above. A halftone should be as true to the original as possible, with respect to each of these characteristics. As has been mentioned, however, tradeoffs usually occur between different halftoning methods.

Power Spectra

Ulichney has shown that much of the quality of a halftone can be measured by calculating its radially-averaged power spectrum. The best halftones all have very similar spectra. This analysis led to the label of "blue-noise halftones." (Ulichney, 1987)

While Ulichney used this method to evaluate halftone images, later researchers developed methods to create

halftones with it. As mentioned before, some blue-noise techniques work by seeking these particular spectral characteristics.

Energy Comparison

Geist, Reynolds, and Suggs (1993) reported that their energy measure algorithm ranked several types of halftones in the same order that they expected most human observers would. They suggested that studies to further test the validity of such energy-based ranking be performed.

For any given image, the minimal energy value (Gibbs measure) corresponds to the halftone with the minimal assumption. To create a halftoning method, they restricted the class of possible halftones by making two assumptions:

> "1. Individual binary pixel values should be strongly correlated with the underlying gray-scale intensities of the individual pixels they represent.
>
> 2. In any small Euclidean neighborhood of pixels, binary pixel values should exhibit a pattern of pairwise correlation (both positive and negative) that allows an accurate representation of the average gray-scale intensity of that

> neighborhood and does so with a minimum of low-frequency noise." (Geist, Reynolds, & Suggs, 1993, p. 143)

Using the proper mathematical representations for these constraints, the halftone with minimal energy becomes the ideal halftone. This energy measure can also, therefore, be used as a relative comparison tool between halftones of the same original image.

Design of the Study

An experiment to test the validity of Geist, Reynolds, and Suggs' energy measure is proposed, based on their suggestion. Halftone images produced by three methods will be used.

Bayer's ordered dither is included in this experiment primarily for two reasons. First, it remains quite popular. Secondly, being a periodic, point-oriented algorithm, it radically differs from the two aperiodic, neighborhood algorithms being used.

While Floyd and Steinberg (1975) first developed error diffusion halftoning using a four-weight filter, later researchers have recommended larger diffusion filters for

better results. Jarvis, Judice, and Ninke's (1976) twelve-weight filter was the first of several large diffusion filters to be developed.

The greater success (and improved blue noise spectra) of Ulichney's random-weighted error diffusion is the basis for its inclusion this experiment. Ulichney wrote concerning his findings,

> "Conventional methods of error diffusion with previously reported error filters were closely examined and found to be fair blue noise generators. Experiments with a broad array of perturbations found that excellent blue noise patterns could be achieved with error filters of four or fewer weights when noise was added and processed on a serpentine raster." (Ulichney, 1987, p. 344)

The test images are meant to collectively represent most types of digital images. They will contain areas of high and low contrast, straight and curved edges, and have small details. These are typically the regions of images that suffer the most from halftoning. Sources of the images will include digitally rendered scenes, scanned photographs,

and some text. Several of the images will be similar to images used by previous researchers.

Methodology

The procedures below will be followed to meet the stated objectives for the study. The results, including all test halftones and their evaluations will be printed. An analysis of the results will lead to a conclusion concerning the validity of energy-based halftone evaluation.

1. Develop software to produce halftones using Ulichney's error diffusion with 50% random weights, Jarvis, Judice, and Ninke's minimal average error (error diffusion), and Bayer's ordered dither.
2. Develop software for the quantitative evaluation of halftones using the energy measure algorithm of Geist, Reynolds, and Suggs.
3. Obtain suitable gray-scale test images, which fit the specified criteria for test variety and difficulty, as mentioned in the Design of the Study.
4. Produce halftones for each gray-scale image using each algorithm from step 1.
5. Compare the resulting halftone images qualitatively. The accuracy of details, edges, contrast, intensity,

and the presence of any artificial patterns or other distortions in each image will be recorded. The evaluation criteria are defined specifically in Appendix B.

6. Compare the halftone images quantitatively, using the energy measure program written in step 2.

7. Correlate the data from steps 5 and 6 to determine the validity of the energy measure.

CHAPTER IV

RESULTS AND ANALYSIS

Obtaining the Data

Each of the images used in the study have been reproduced in Appendix C. Five different gray-scale pictures were halftoned three different ways, yielding fifteen halftones. For each of the five, a higher resolution prescaled halftone is also printed, to better approximate the original gray-scale image. Therefore, a total of twenty images are shown.

The complete program code for the energy calculations and the halftones used in the study is listed in Appendix D. All code was written in ANSI C, and compiled and executed on an IBM RS/6000 Model 580 running AIX version 3.2.5.

Results

The results of the visual evaluations are given in Table I (See Appendix B for explanations of the specific rating criteria used). The average (arithmetic mean) score for each of the fifteen images was computed and also appears

in Table I. For these values, greater numbers indicate better performance.

Results from the application of Geist, Reynolds, and Suggs' energy measure to the images are shown in Table II. All values shown are negative as expected, and smaller values indicate better performance.

Table I. Visual Evaluation Results

Bayer Ordered Dither	Image #1	Image #2	Image #3	Image #4	Image #5
Detail/Edges:	70	65	40	85	40
Intensity/Contrast:	75	85	85	85	74
Artifacts/Patterns:	50	50	50	50	35
Shape Distortion:	95	70	70	90	75
Average Score:	73	68	61	78	56

Jarvis, Judice, & Ninke Filter	Image #1	Image #2	Image #3	Image #4	Image #5
Detail/Edges:	90	85	75	85	90
Intensity/Contrast:	65	90	75	85	95
Artifacts/Patterns:	65	70	65	70	60
Shape Distortion:	50	70	70	90	85
Average Score:	68	79	71	83	83

50% Random Floyd-Steinberg Filter	Image #1	Image #2	Image #3	Image #4	Image #5
Detail/Edges:	95	80	90	80	80
Intensity/Contrast:	90	90	95	95	95
Artifacts/Patterns:	90	70	95	90	95
Shape Distortion:	70	90	85	85	85
Average Score:	86	83	91	88	89

Interpretation of the Data

One difficulty arises when interpreting results of the energy measure. The range of values obtainable using the energy calculation is dependent on several variables. It depends on the size of the image, and the content of the original gray-scale image, as well as the halftone being measured. For this reason, the energy values are only significant for use as a relative comparison between two or more halftones of the same original image. All of the images evaluated in this study were of identical size to remove the variable of size.

Using linear regression to determine the correlation between the fifteen quality ratings and their respective energies yields a correlation coefficient, $r = -0.0155$. This small a number indicates that no correlation exists at all.

To remove the effects of the different spectral characteristics of each original image, the energy values were normalized. These figures are shown in Table III. Each was normalized by dividing them by the absolute value of the average of the energy values for the three halftones of each image. This is a simple approximation, but allows the energy

measure to be used loosely for comparisons of different images.

Table IV summarizes the two sets of data extracted from Table I and Table III. Using the same linear regression as before, but comparing the visual scores to the normalized energy values shown in Table IV yields a correlation of -0.606. This indicates a strong but non-conclusive relationship does exist. The correlation is negative, as it should be, since better images are supposed to produce greater negative values.

The fifteen samples can also be viewed another way. By computing the average results for each halftoning method used (each row in Table IV), three artificial samples are obtained. These figures give an overall idea of the performance of each halftoning method, as measured by visual inspection versus image energy. These average values are shown in Table V. Using linear regression again, the correlation between these figures is a very strong -0.936.

Table II. Energy Measure Results

Energy Value	Image #1	Image #2	Image #3	Image #4	Image #5
Bayer Ordered Dither	-51988.17	-59247.36	-63774.08	-51922.49	-48631.70
Jarvis, Judice, & Ninke Filter	-51789.48	-65085.64	-64508.38	-52488.24	-49161.00
50% Random Floyd-Steinberg Filter	-52893.55	-62835.74	-65902.60	-53102.55	-49828.81

Note: Smaller values indicate better performance.

Table III. Normalized Energy Measure Results

Energy Value	Image #1	Image #2	Image #3	Image #4	Image #5
Bayer Ordered Dither	-0.9955	-0.9496	-0.9853	-0.9889	-0.9883
Jarvis, Judice, & Ninke Filter	-0.9917	-1.0432	-0.9966	-0.9997	-0.9991
50% Random Floyd-Steinberg Filter	-1.0128	-1.0072	-1.0181	-1.0114	-1.0126

Note: Smaller values indicate better performance.

Table IV. Evaluation Summary by Image

Visual Rank vs. **Norm. Energy**	Image #1	Image #2	Image #3	Image #4	Image #5
Bayer Ordered Dither	73 -0.9955	68 -0.9496	61 -0.9853	78 -0.9889	56 -0.9883
Jarvis, Judice, & Ninke Filter	68 -0.9917	79 -1.0432	71 -0.9966	83 -0.9997	83 -0.9991
50% Random Floyd-Steinberg Filter	86 -1.0128	83 -1.0072	91 -1.0181	88 -1.0114	89 -1.0126

Table V. Averages of Results by Halftone Method

	Visual Evaluation	**Average Normalized Energy**
Bayer Ordered Dither	66.95	-0.9815
Jarvis, Judice, & Ninke Filter	76.50	-1.0061
50% Random Floyd-Steinberg Filter	87.25	-1.0124

CHAPTER V

SUMMARY AND CONCLUSIONS

Summary of Results

The first objective was to qualitatively compare five images produced by each of three methods: Bayer ordered dither; Jarvis, Judice, and Ninke's minimal average error; and Ulichney's 50%-random-weighted error diffusion. Also the accuracy of image details, edges, contrast, intensity, and the presence of any artificial patterns or other distortions in each image were to be rated numerically on a checklist (See Appendix B for an example checklist and the specific criteria).

To meet Objective 1, several steps were necessary. Software was written to create three types of halftones as planned. Five grayscale images were each halftoned using the software, yielding fifteen test images. The quality of these halftones were compared by inspection using the criteria listed, and results were recorded in Table I.

The second objective was to evaluate the same halftone images quantitatively using the energy measure algorithm of

Geist, Reynolds, and Suggs. To satisfy this objective, the software developed for the first objective was modified to also calculate image energy. All program code is listed in Appendix D and the energy algorithm is given in Appendix E.

The third and final objective was to correlate the averaged qualitative results with the energy calculations to determine the validity of the energy measure. This was performed by applying linear regression to the two sets of data (qualitative rankings and energy values). A method of normalizing the data was also used in the analysis to determine the success of the energy measure algorithm.

Conclusions from the Study

The energy measure, of course, cannot produce halftone quality rankings that will agree with everyone's subjective opinions concerning a given image. It is clear, however, that the energy measure introduced by Geist, Reynolds, and Suggs successfully ranks images based on their gray-scale accuracy, their detail, and their "blue-noise" spectra. Most recent digital halftoning research seems to agree that these are primary characteristics of quality halftones (Ulichney, 1987; Peli, 1991; Mitsa & Parker, 1992; Scheermesser, Broja, & Bryngdahl, 1993)

The particular halftone algorithms used in the study were chosen in part, to aid and support the visual comparisons. One would ordinarily expect that random-weighted error diffusion should outperform standard error diffusion, which should likewise outperform the Bayer ordered dither (Ulichney, 1987; Jarvis, Judice, & Ninke, 1976). The results for image #1, a gray-scale ramp, shown in Figures 12-15 are notable, in that they differ from this expectation. The Bayer dither in Figure 13 received a second-place ranking in the visual comparison, since it seems to do a better job of representing the original than the Jarvis, Judice, and Ninke filter in Figure 14. Interestingly, the energy measure also awards Figure 13 a better score than Figure 14.

The energy measure has been found to be a valid, useful tool for the evaluation of binary digital halftone quality. The final correlation coefficient of -0.936 indicates a very strong link between image energy and image quality. (A coefficient of ±1.0 would indicate the absolute dependence of energy on quality, while a coefficient of 0 would indicate no relationship.)

CHAPTER VI

RECOMMENDATIONS

Having established that the energy measure can be very useful as a quantitative halftone evaluation tool, two things remain which could greatly enhance its usability in practice. First and most importantly, an accurate method of standardizing energy values for arbitrary images needs to be developed. Secondly, the processing requirements for calculating image energy must be reduced before it will find popularity.

The energy values produced by Geist, Reynolds, and Suggs' algorithm do not have a constant range, so a single halftone and its energy value are meaningless without other halftones of the same image to use for comparison. While an average of several halftone algorithms' energy values can be used to normalize the energy for a given image, as was done in this study, a better solution is needed. A more accurate method of standardizing the energy values, which does not require computing additional halftones, should be developed.

The other difficulty with the application of Geist, Reynolds, and Suggs' energy measure is its computational

requirements. When compared to the complexity of most halftoning methods, its storage and processing requirements are exceedingly large. Efforts should be made to reduce the computational complexity of the algorithm, while retaining its accuracy.

BIBLIOGRAPHY

Bayer, B. (1973). An Optimum Method for Two Level Rendition of Continuous-Tone Pictures. Proceedings of the IEEE International Conference on Communications, New York: IEEE, 26-11 - 26-15.

Floyd, R. & Steinberg, L. (1975). An Adaptive Algorithm for Spatial Gray Scale. SID 1975, International Symposium on Digital Technology, Papers, 36-37.

Geist, R., Reynolds, R., & Suggs, D. (1993). A Markovian Framework for Digital Halftoning. ACM Transactions on Graphics, 12(2), 136-159.

Gentile, R. S., Walowit, E., & Allebach, J. P. (1990). Quantization and Multilevel Halftoning of Color Images for Near-Original Image Quality. Journal of the Optical Society of America, 7, 1019-1026.

Jarvis, J. F., Judice, C. N., & Ninke, W. H., (1976). A Survey of Techniques for the Display of Continuous Tone Pictures on Bilevel Displays. Computer Graphics and Image Processing, 5, 13-40.

Knuth, D. (1987). Digital Halftones by Dot Diffusion. ACM Transactions on Graphics, 6, 245-273.

Linotype-Hell Company, (January 1993). Resolution, Screen Ruling and Halftone Dot Shape. Technical Guidebook, A9, Hauppage, NY: Author.

Mitsa, T., & Parker, K. (1992). Digital Halftoning Technique Using a Blue-Noise Mask. Journal of the Optical Society of America, 9, 1920-1929.

Peli, E. (1991). Multiresolution, Error-Convergence Halftone Algorithm. Journal of the Optical Society of America, 8, 625-636.

Rimmer, S. (1993). Windows Bitmapped Graphics. Windcrest/ McGraw-Hill.

Rogers, David F., (1985). Procedural Elements for Computer Graphics. New York: McGraw-Hill.

Scheermesser, T., Broja, M., & Bryngdahl, O. (1993). Adaptation of Spectral Constraints to Electronically Halftoned Pictures. Journal of the Optical Society of America, 10, 412-417.

Stevenson, R. & Arce, G. (1985). Binary Display of Hexagonally Sampled Continuous-Tone Images. Journal of the Optical Society of America, 2, 1009-1013.

Stucki, P. (1979). Image Processing for Document Reproduction. In P. Stucki (Ed.), Advances in Digital Image Processing. New York: Plenum. 177-218.

Ulichney, R., (1987). Digital Halftoning. Cambridge, MA: MIT Press.

Weissbach, S., & Wyrowski, F. (1992). Error Diffusion Procedure: Theory and Applications in Optical Signal Processing. Applied Optics, 31(14), 2518-2534.

Related Literature

Bellanger, M. (1986). Digital Processing of Signals: Theory and Practice. Chichester: John Wiley & Sons.

Booth, K., Bryden, M., Cowan, W., Morgan, M., & Plante, B. (1987, September). On the Parameters of Human Visual Performance: An Investigation of the Benefits of Antialiasing. IEEE Computer Graphics and Applications, 7, 34-41.

Crow, F. (1978). The Use of Grayscale for Improved Raster Display of Vectors and Characters. Computer Graphics, 12(3), 1-5.

Dayhoff, J. (1990). Neural Network Architectures: An Introduction. New York: Van Nostrand Reinhold.

Earnshaw, R. A. (Ed.). (1985). Fundamental Algorithms for Computer Graphics (NATO ASI Series. Series F, Computer and System Sciences; 17). Berlin: Springer-Verlag.

Gupta, S., & Sproull, R. (1981). Filtering Edges for Gray-Scale Displays. Computer Graphics, 15(3), 1-5.

Hamming, R. W. (1983). Digital Filters (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Kinderman, R., & Snell, J. (1980). Markov Random Fields and their Applications. Providence, Rhode Island: American Mathematical Society.

Simpson, P. (1990). Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations. New York: Pergamon Press.

Stanley, W. D. (1975). Digital Signal Processing. Reston, VA: Reston Publishing.

APPENDIX

APPENDIX A

Related Procedures in Image Processing

Several image processing techniques have been mentioned in passing, which can significantly affect halftone quality. Especially when halftoning with bilevel displays, tone scale adjustment and sharpening may be necessary to achieve the quality of results desired. Prescaling, if possible, can dramatically improve halftones.

Tone Scale Adjustment

When halftoning gray-scale images with just black and white, it is customary to remap, or adjust, the intensities in the image to better preserve the brightness and contrast of the image. Tone scale adjustment can also add contrast at very light and dark regions, enlivening otherwise "flat" images (Ulichney, 1987, pp. 11-14; Rimmer, 1993, pp. 331-333).

Sharpening

Sharpening involves increasing the contrast of an image on edges within the image. Sharpening is also called "edge-enhancement." Some halftoning algorithms cause a degree of

sharpening as a side-effect. In particular, error diffusion filters with many weights tend to sharpen edges. For example, compare the twelve-weight filters in Figure 7 and Figure 8 to the four-weight filter in Figure 5.

While some amount of sharpening can enhance a halftone, it depends on the particular image. Ulichney argues that sharpening should be separately controlled from halftoning, not inherent in the process. He recommends sharpening the image prior to halftoning (Ulichney, 1987, p. 253).

Prescaling

Prescaling is a process which can significantly improve the results of halftoning. Prescaling is performed by scaling the image resolution up by some factor before halftoning. Prescaling is particularly successful when an error-diffusion technique is employed. The primary drawback to prescaling is its memory and computation requirements. Since the resolution is increased by the square of the scaling factor, so does the memory size of the image, and the time needed to process it (Rimmer, 1993, pp. 339-340).

Obviously, prescaling will require that the halftoned image is larger than the original, unless it can be printed

at a higher resolution. Figure 1, for example, was prescaled by a factor of four in both dimensions, before it was halftoned. Since it was printed at 300 dpi, it is still the same size as the other figures, printed at 75 dpi.

APPENDIX B

Evaluation Criteria and Data Sheet

Each image will be evaluated on several specified criteria, using a scale of 0 to 100, where 0 is the worst performance and 100 is the most accurate. The criteria are defined for this study as follows:

Detail/Edges: Are small features in the original still discernible? If there are letters, are they legible? Jarvis, Judice, and Ninke (1976, p. 30) define edge emphasis as "creating an enhanced legibility of textual, line and other material of high detail."

Intensity/Contrast: Do local intensities closely match the original gray levels? Halftoning techniques are meant to give the subjective appearance of continuous tone (Jarvis, Judice, & Ninke, 1976, p. 13).

Artifacts/Patterns: Are there artificial patterns which should not be present? Image artifacts have been a primary problem and reason for many developments in halftoning. (Ulichney, 1987; Jarvis, Judice, & Ninke, 1976, p. 27, 31; Knuth, 1987, p. 246)

Distortion: Are any shapes in the image obviously distorted? Shape distortion by the diffusion process has been called "directional hysteresis" by Ulichney (1987, p. 253). This distortion is most evident in regions of slowly-varying intensity.

Average Score: The arithmetic mean of the four ratings above. This result will be used to calculate the correlation with image energy.

Halftone Evaluation Data Sheet

Bayer Dither Test Images

	#1	#2	#3	#4	#5
Detail/Edges:	_____	_____	_____	_____	_____
Intensity/Contrast:	_____	_____	_____	_____	_____
Artifacts/Patterns:	_____	_____	_____	_____	_____
Shape Distortion:	_____	_____	_____	_____	_____
Average Score:	_____	_____	_____	_____	_____

Jarvis, Judice, & Ninke Test Images

	#1	#2	#3	#4	#5
Detail/Edges:	_____	_____	_____	_____	_____
Intensity/Contrast:	_____	_____	_____	_____	_____
Artifacts/Patterns:	_____	_____	_____	_____	_____
Shape Distortion:	_____	_____	_____	_____	_____
Average Score:	_____	_____	_____	_____	_____

Random-Weight Error Diffusion Test Images

	#1	#2	#3	#4	#5
Detail/Edges:	_____	_____	_____	_____	_____
Intensity/Contrast:	_____	_____	_____	_____	_____
Artifacts/Patterns:	_____	_____	_____	_____	_____
Shape Distortion:	_____	_____	_____	_____	_____
Average Score:	_____	_____	_____	_____	_____

APPENDIX C

Test Images

The five original gray-scale images used in the study were obtained from several sources. The binary halftones of those gray-scale images were all produced by programs written for this study. The source code is printed in Appendix D.

Image #1 is a series of horizontal ramps, containing the entire range of 256 grays from black to white. Similar images have been used to evaluate halftones by Ulichney (1987), Peli (1991), and Jarvis, Judice, and Ninke (1976).

Image #2 is a computer-generated ray-traced image. It very closely resembles one used by Geist, Reynolds, and Suggs (1993). It was created using the Persistence of Vision (POV-Ray) ray tracing program.

Image #3 has a vertical grayscale ramp for background, with several lines of text superimposed. The text is solid black at the top, 50% gray in the center, and solid white at the bottom.

Image #4 was digitized from a photograph. It is a NASA picture of a space shuttle launch, and was retrieved from a public NASA image repository on Internet.

Image #5 is the same image of San Francisco's Golden Gate Bridge that was used to illustrate digital halftoning methods earlier in this paper. It was also downloaded from a public site on Internet.

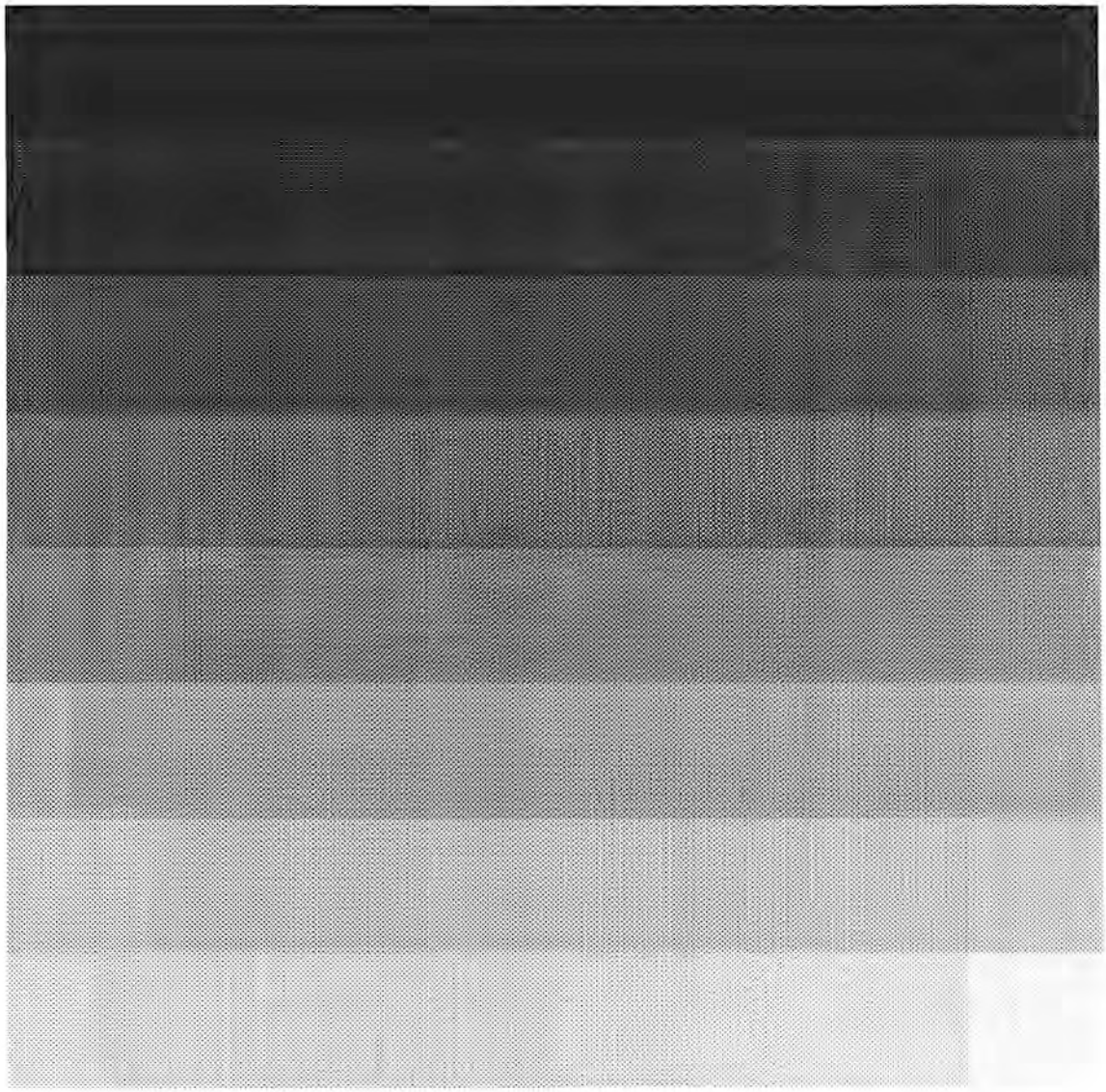

Figure 12. Image #1. Prescaled Halftone for Reference. (300 dpi)

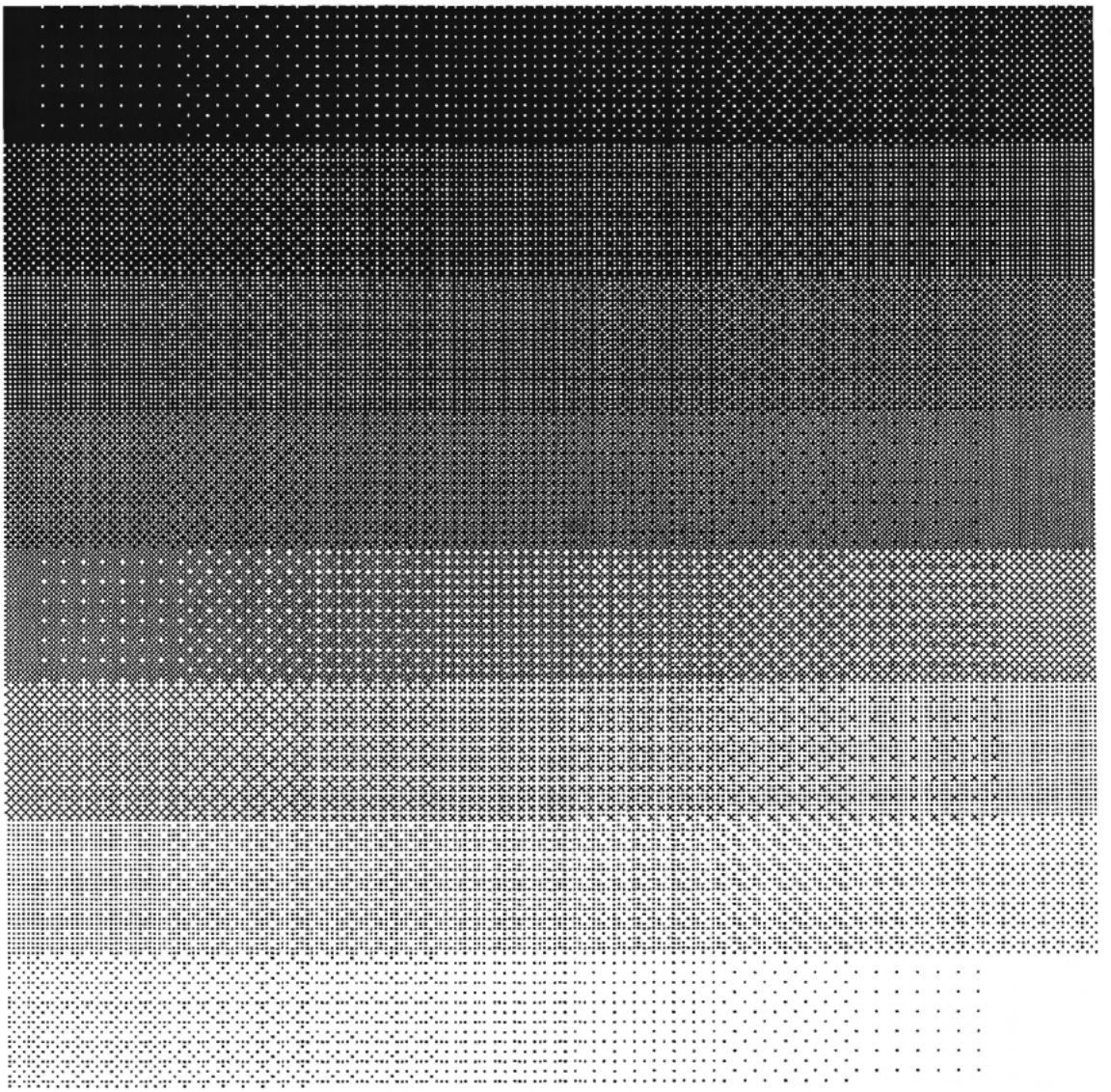

Figure 13. Image #1. Bayer Ordered Dither. (75 dpi)

U = -51988.17

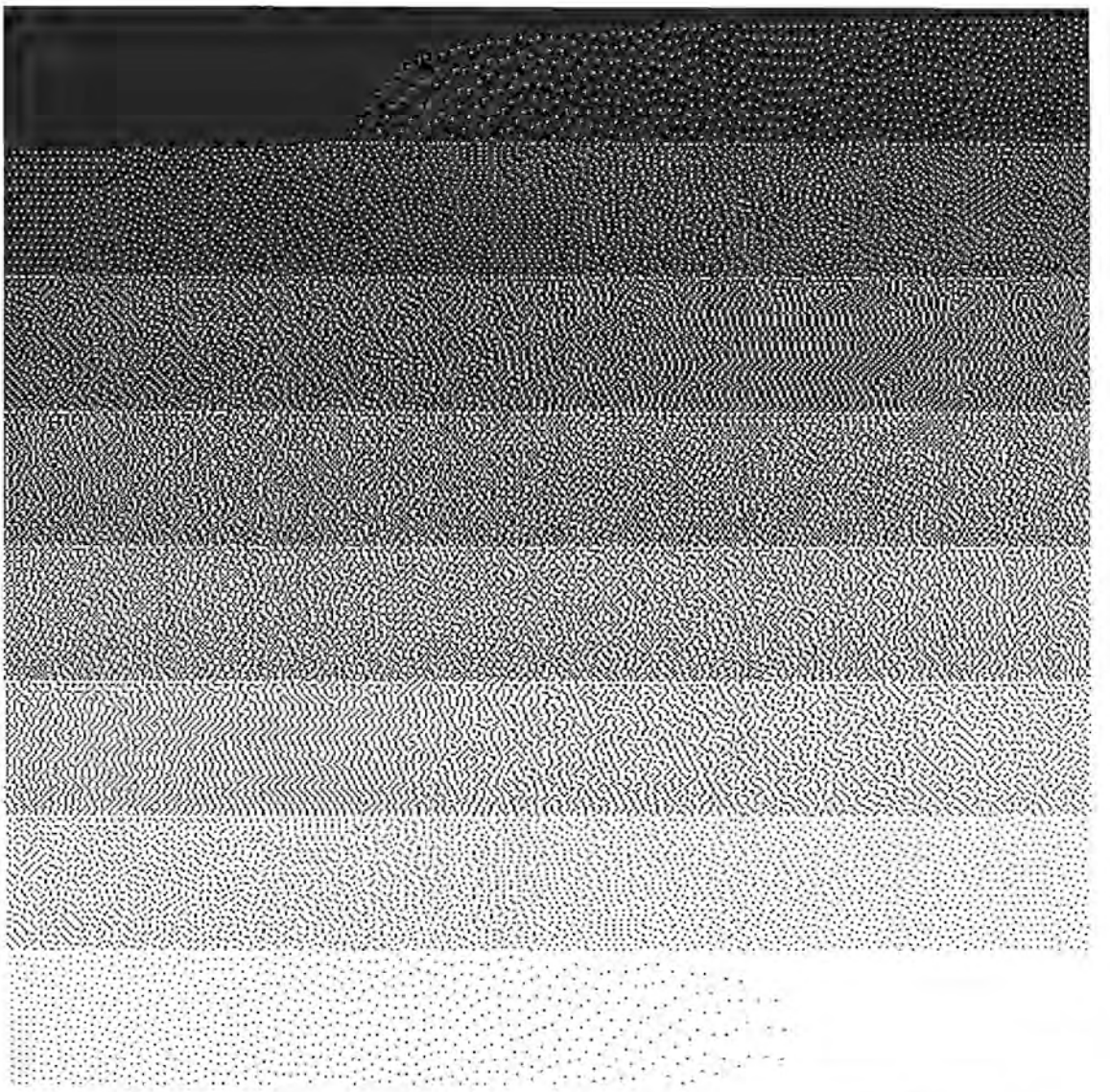

Figure 14. Image #1. Jarvis, Judice, & Ninke Filter. (75 dpi)

U = -51789.48

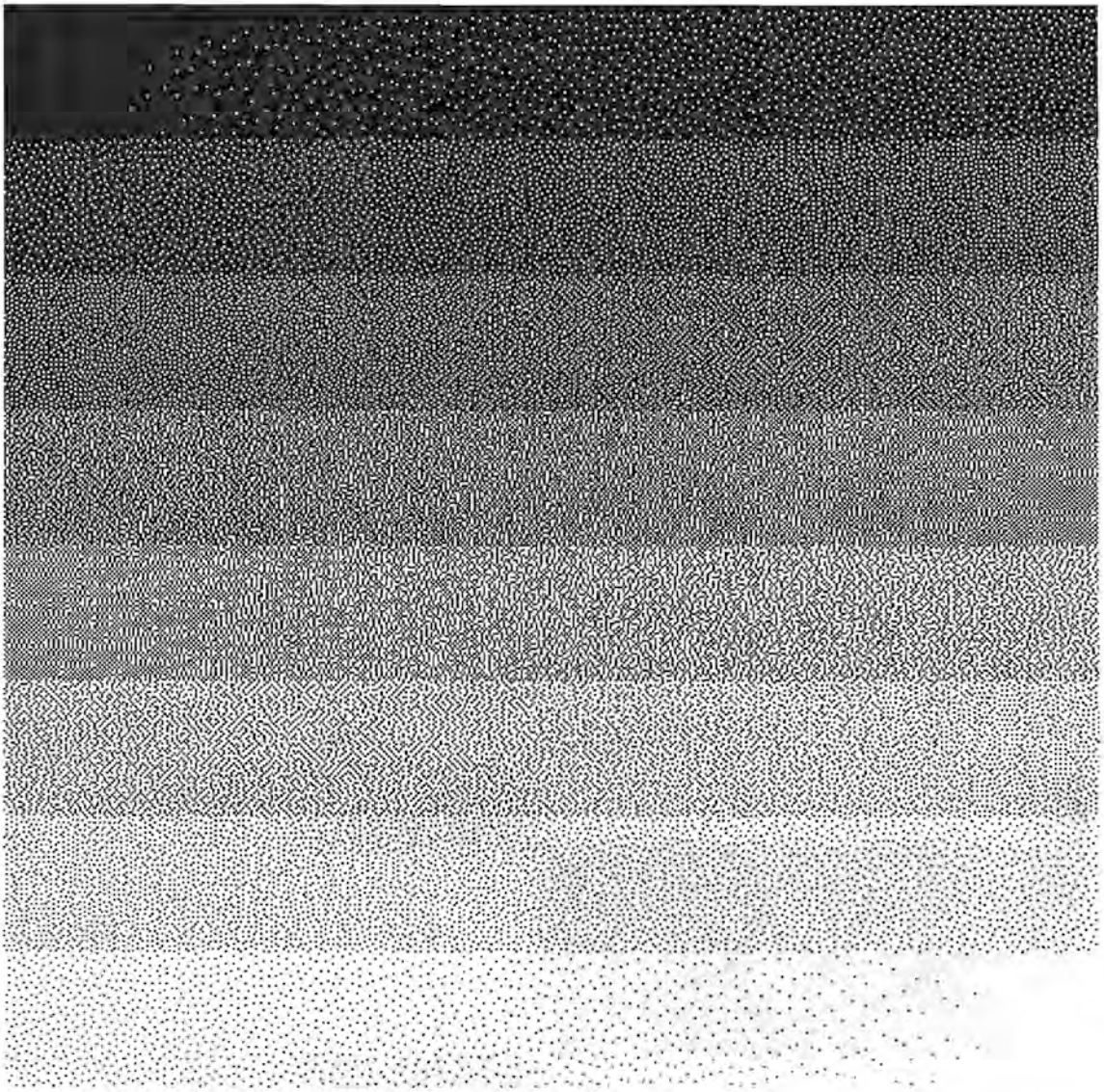

Figure 15. Image #1. 50-Percent Random Floyd-Steinberg Filter. (75 dpi)

U = -52893.55

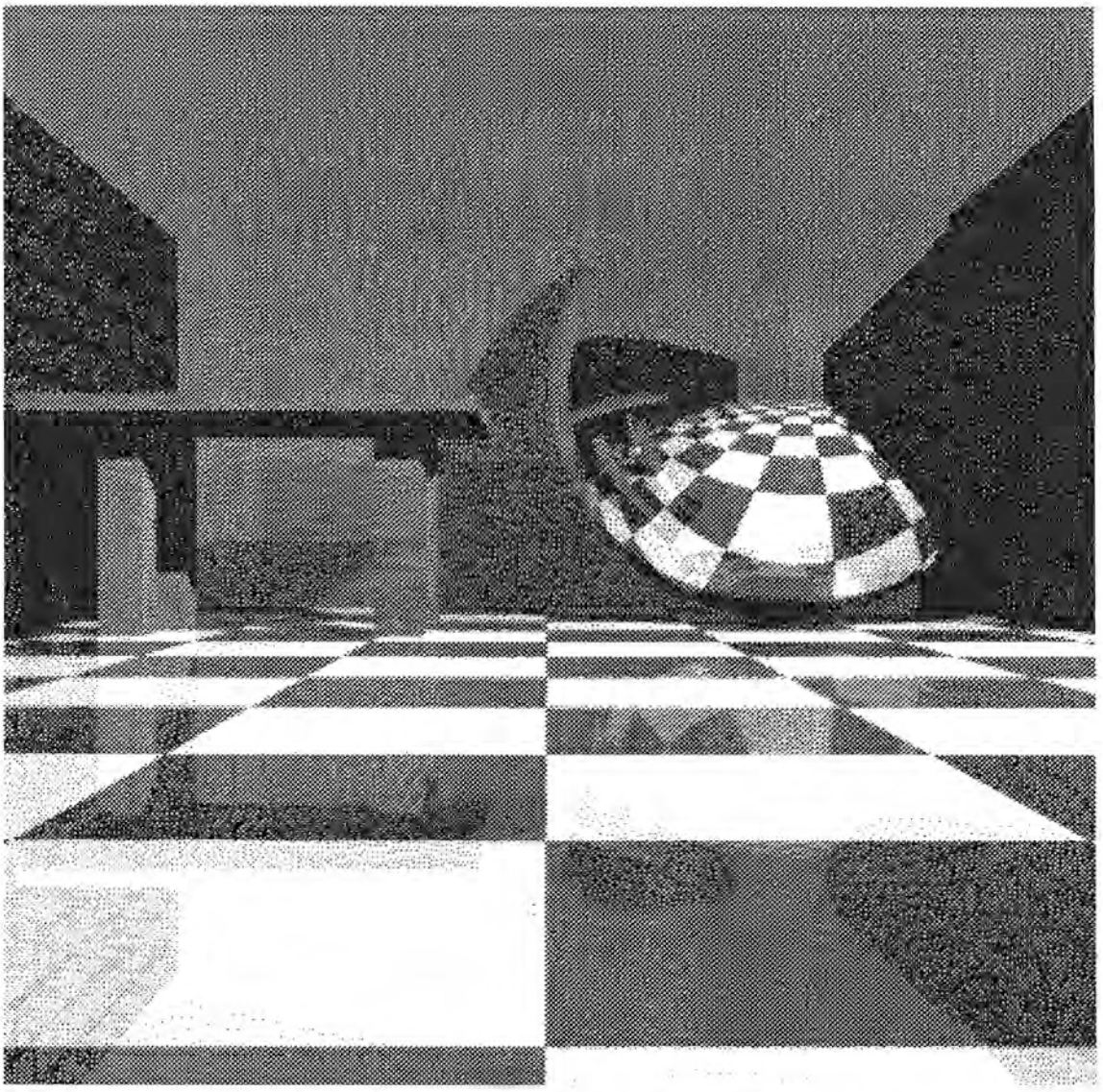

Figure 16. Image #2. Prescaled Halftone for Reference. (300 dpi)

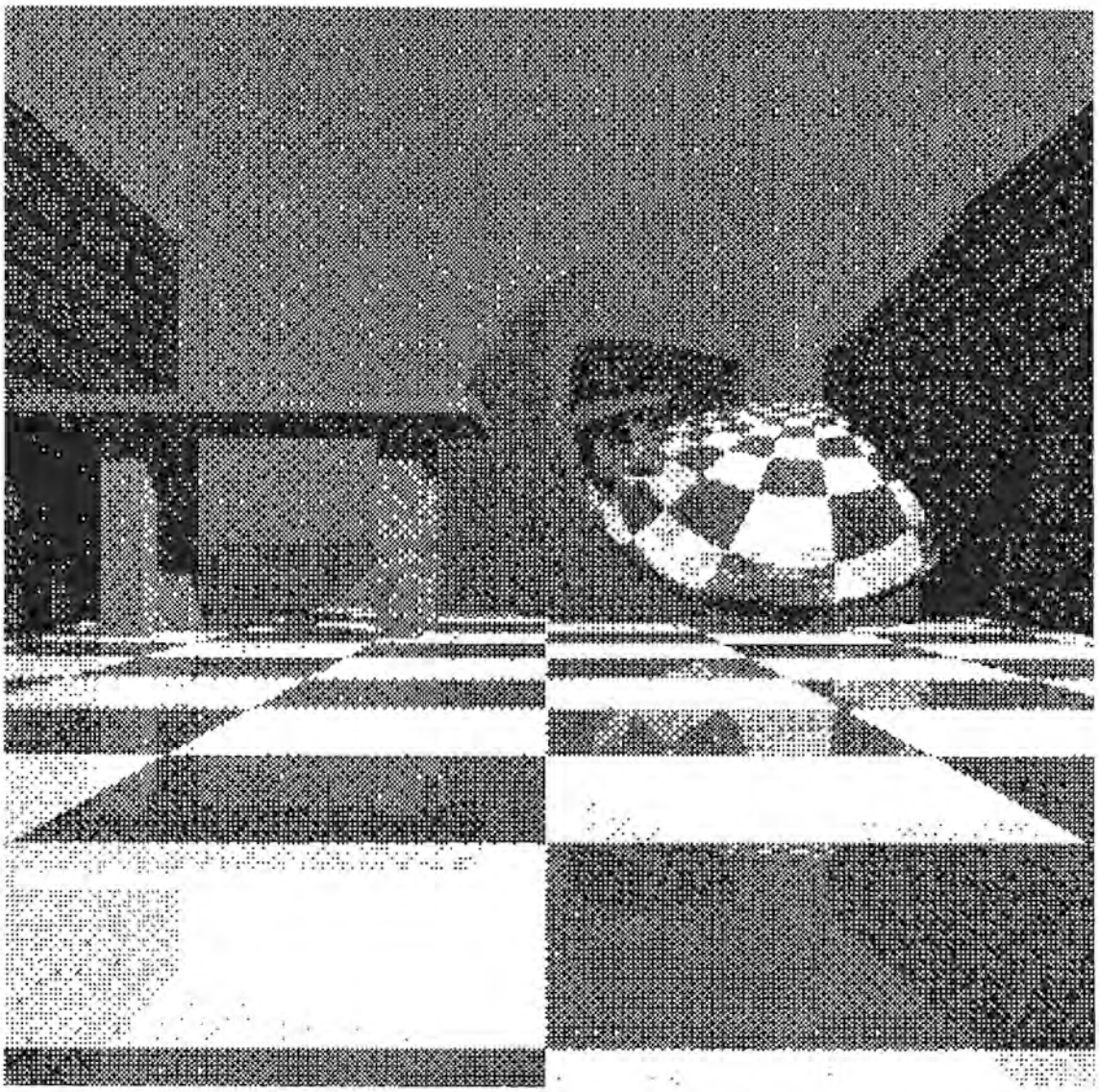

Figure 17. Image #2. Bayer Ordered Dither. (75 dpi)

U = -59247.36

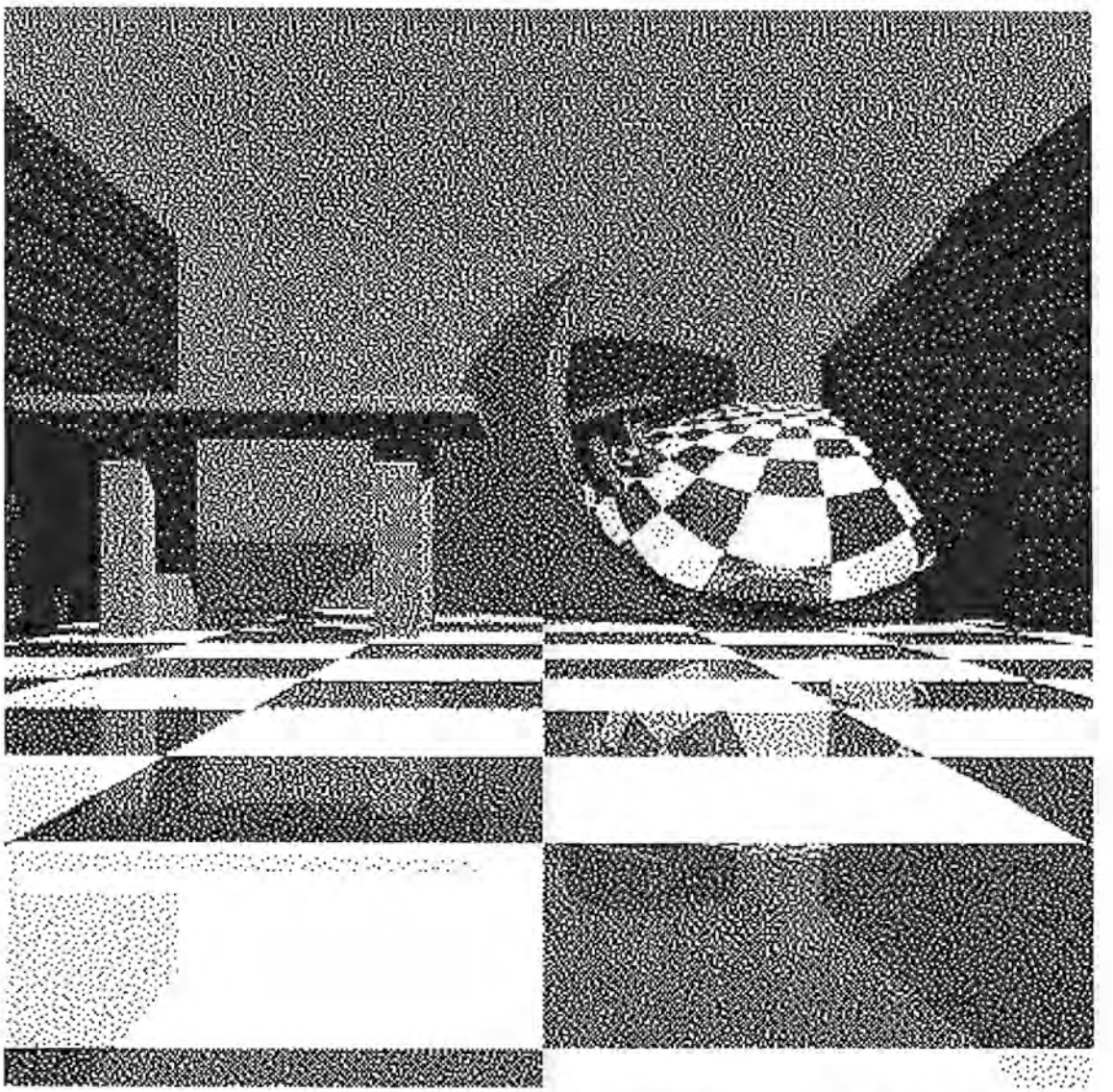

Figure 18. Image #2. Jarvis, Judice, & Ninke Filter. (75 dpi)

U = -65085.64

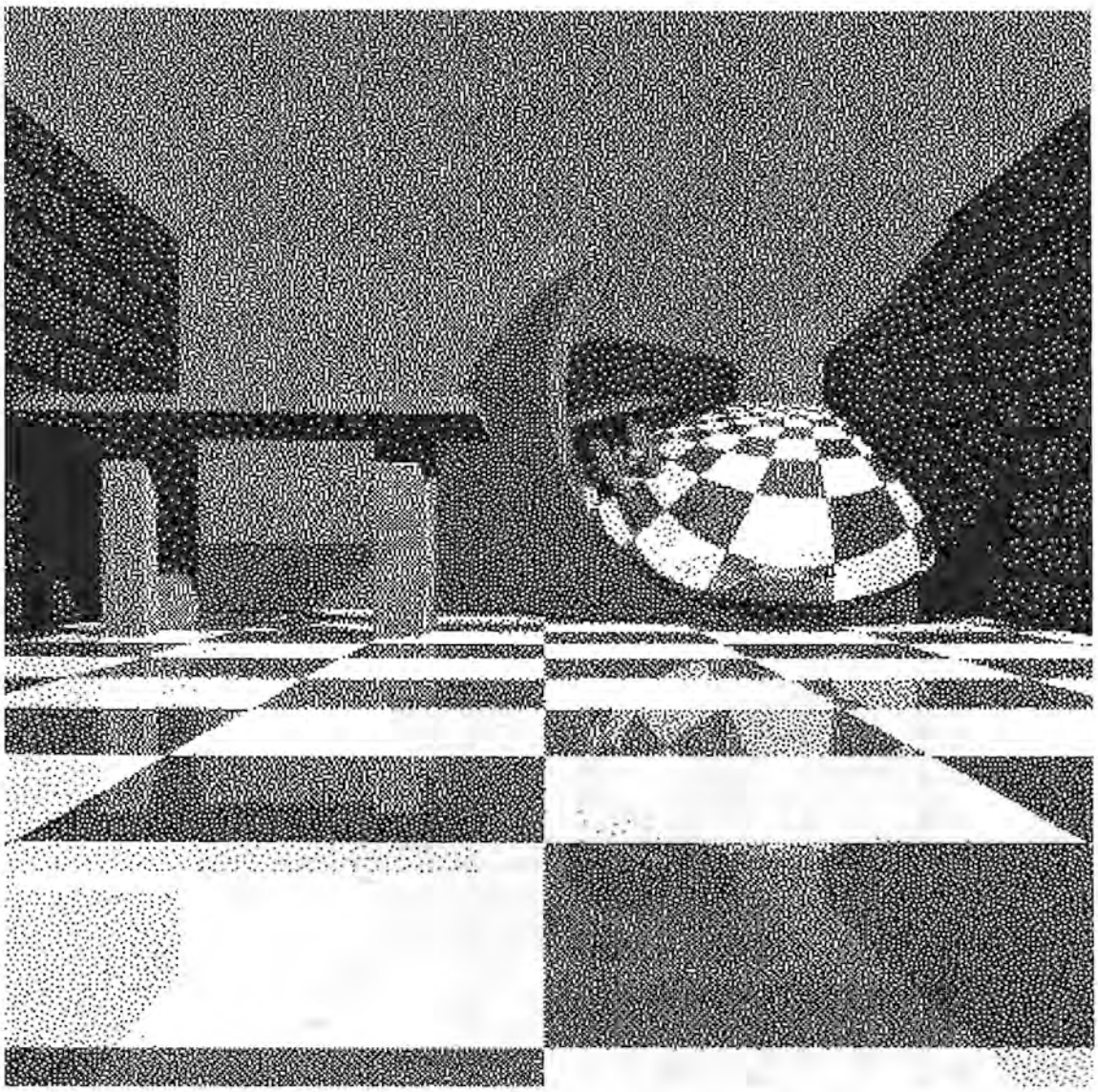

Figure 19. Image #2. 50-Percent Random Floyd-Steinberg Filter. (75 dpi)

U = -62835.74

Figure 20. Image #3. Prescaled Halftone for Reference. (300 dpi)

Figure 21. Image #3. Bayer Ordered Dither. (75 dpi)

U = -63774.08

Figure 22. Image #3. Jarvis, Judice, & Ninke Filter. (75 dpi)

U = -64508.38

Figure 23. Image #3. 50-Percent Random Floyd-Steinberg Filter. (75 dpi)

U = -65092.60

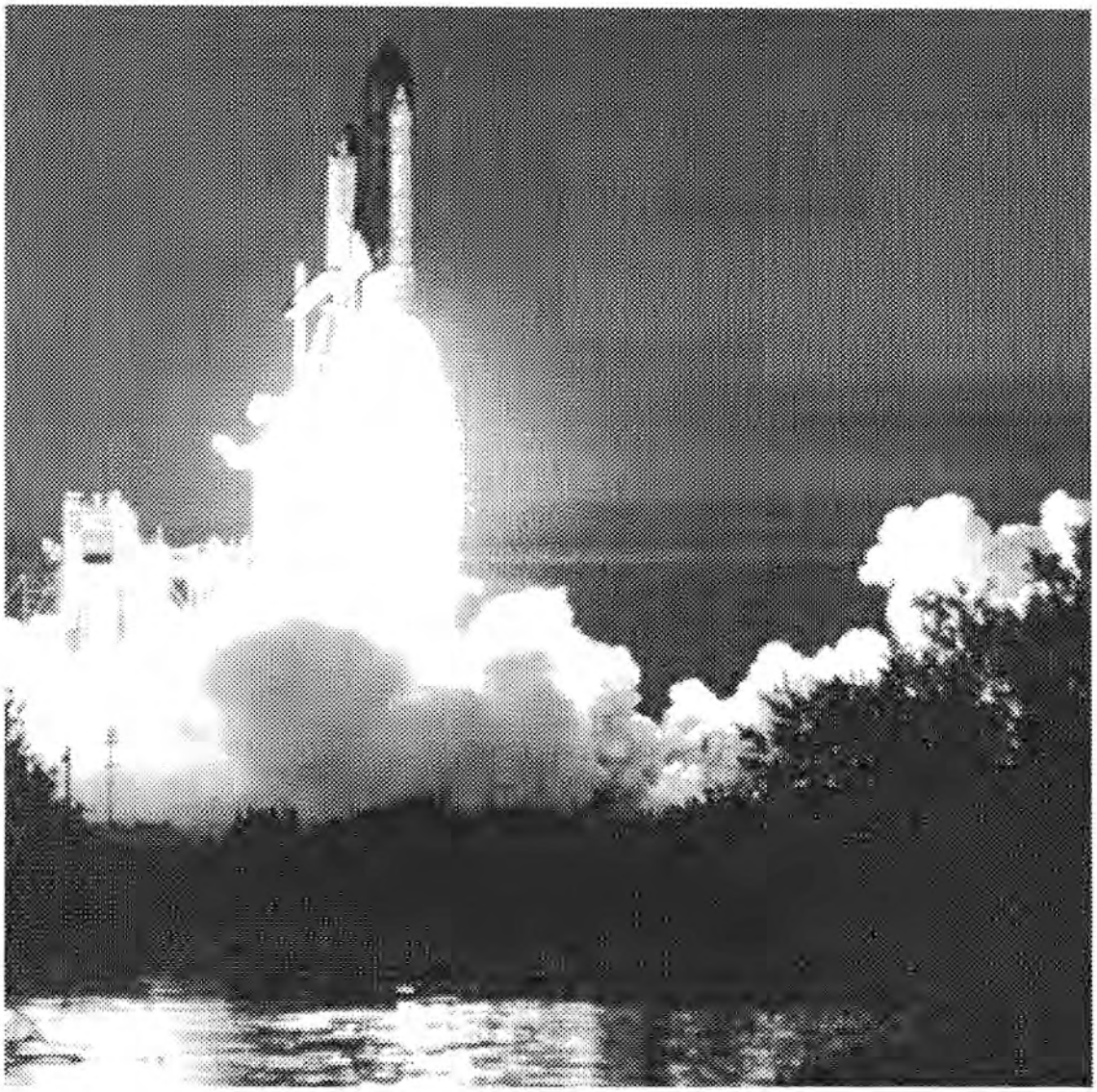

Figure 24. Image #4. Prescaled Halftone for Reference. (300 dpi)

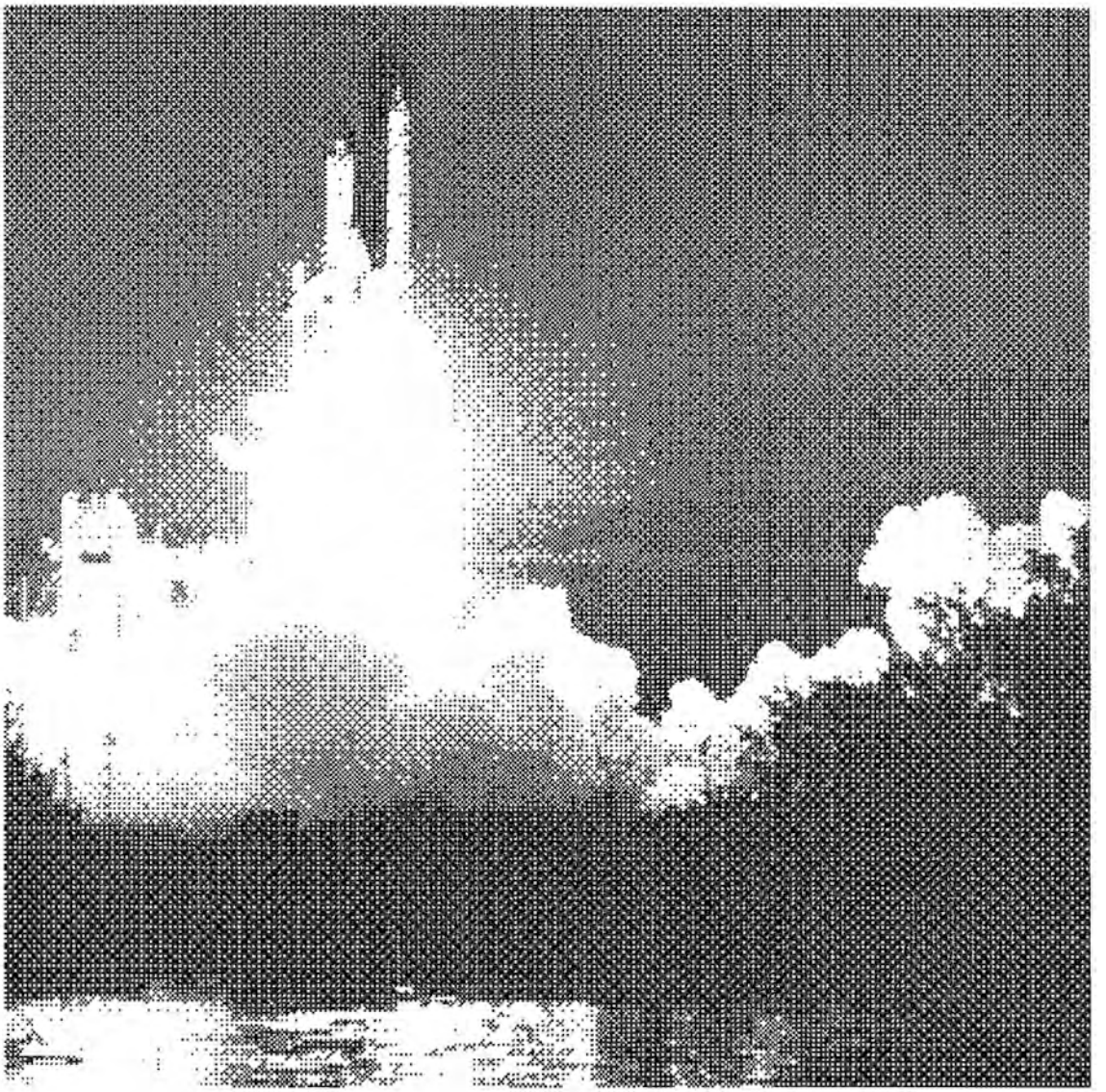

Figure 25. Image #4. Bayer Ordered Dither. (75 dpi)

U = -51922.49

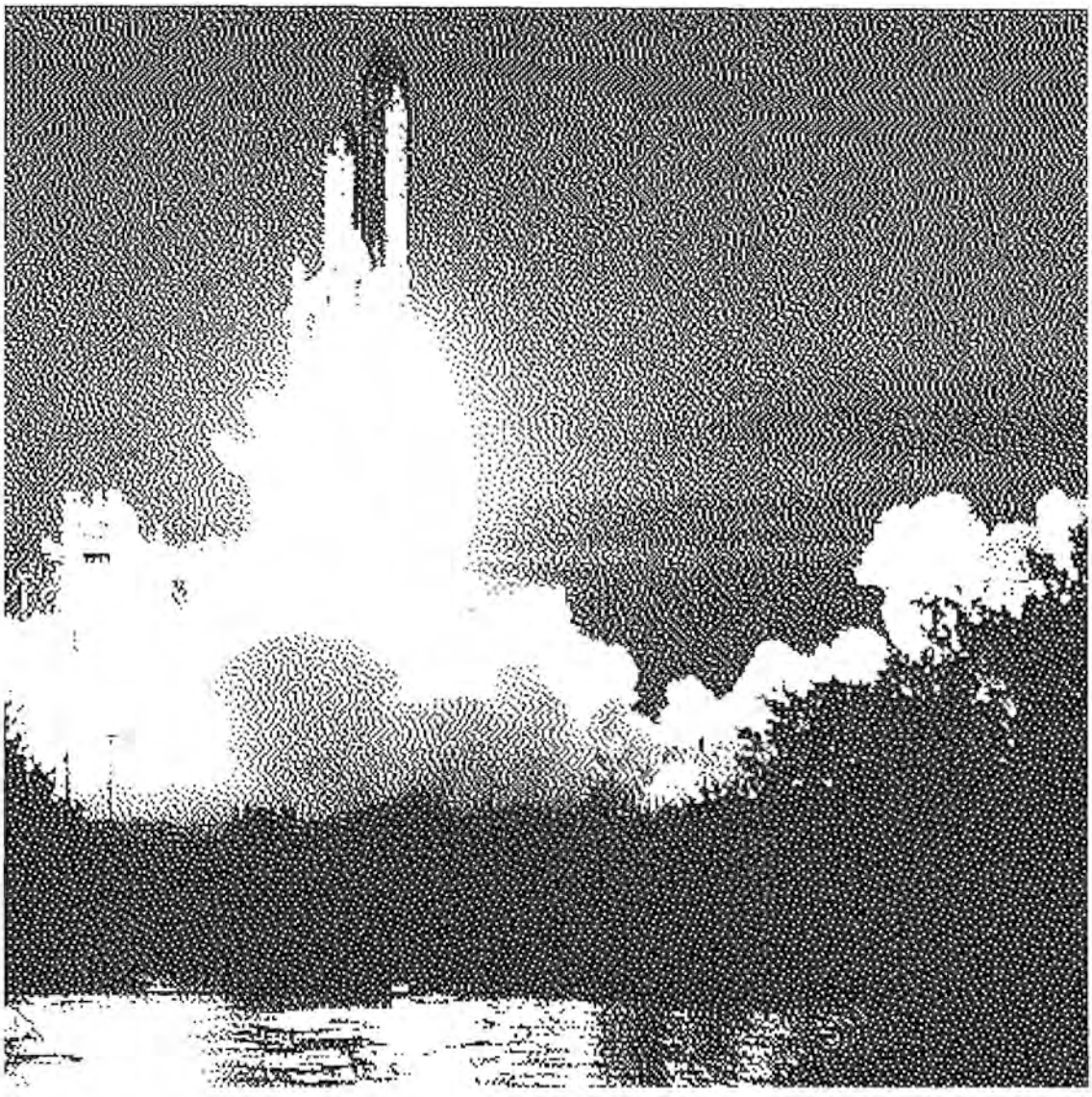

Figure 26. Image #4. Jarvis, Judice, & Ninke Filter. (75 dpi)

$$U = -52488.24$$

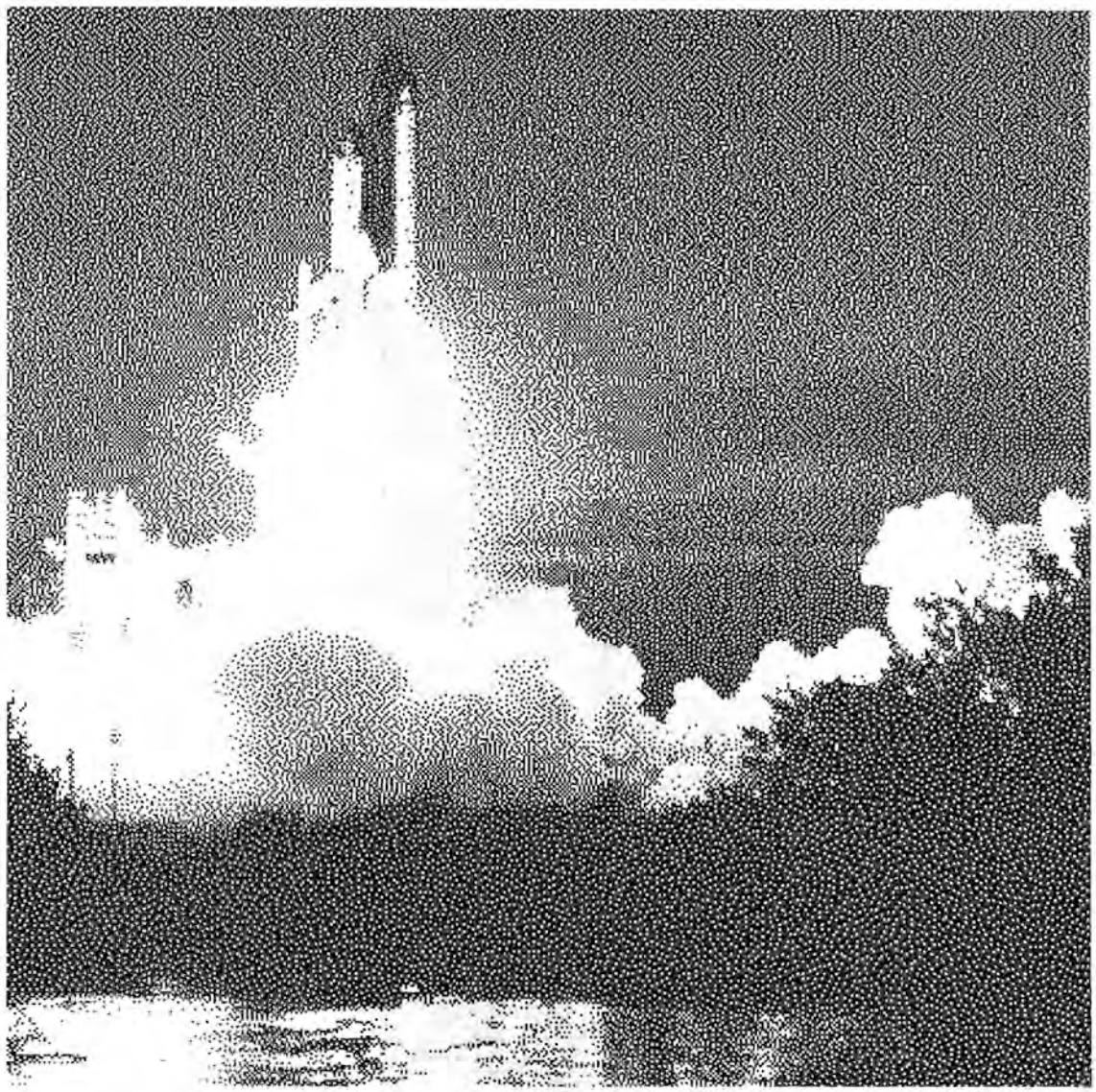

Figure 27. Image #4. 50-Percent Random Floyd-Steinberg Filter. (75 dpi)

U = -53102.55

Figure 28. Image #5. Prescaled Halftone for Reference. (300 dpi)

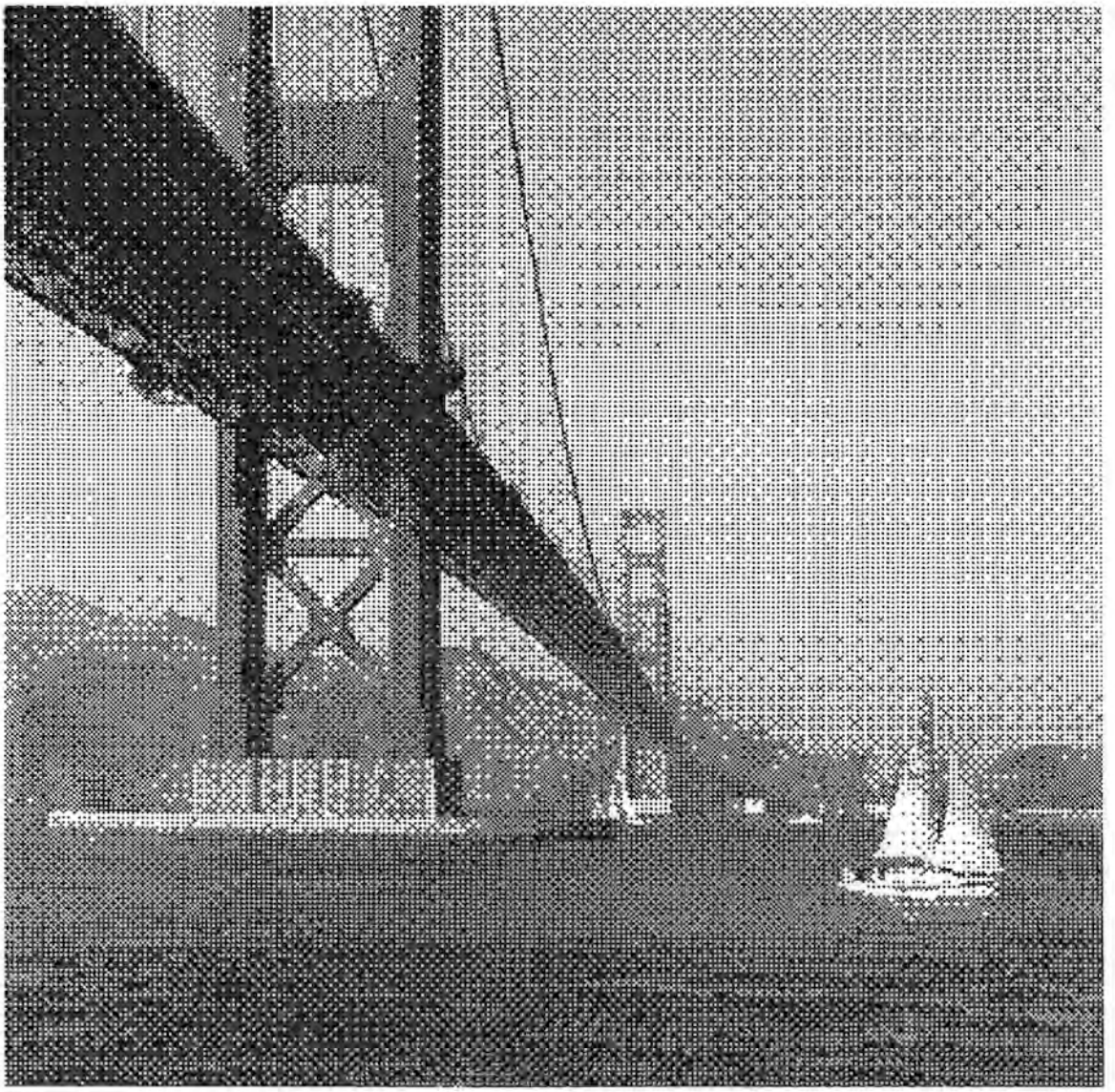

Figure 29. Image #5. Bayer Ordered Dither. (75 dpi)

U = -48631.70

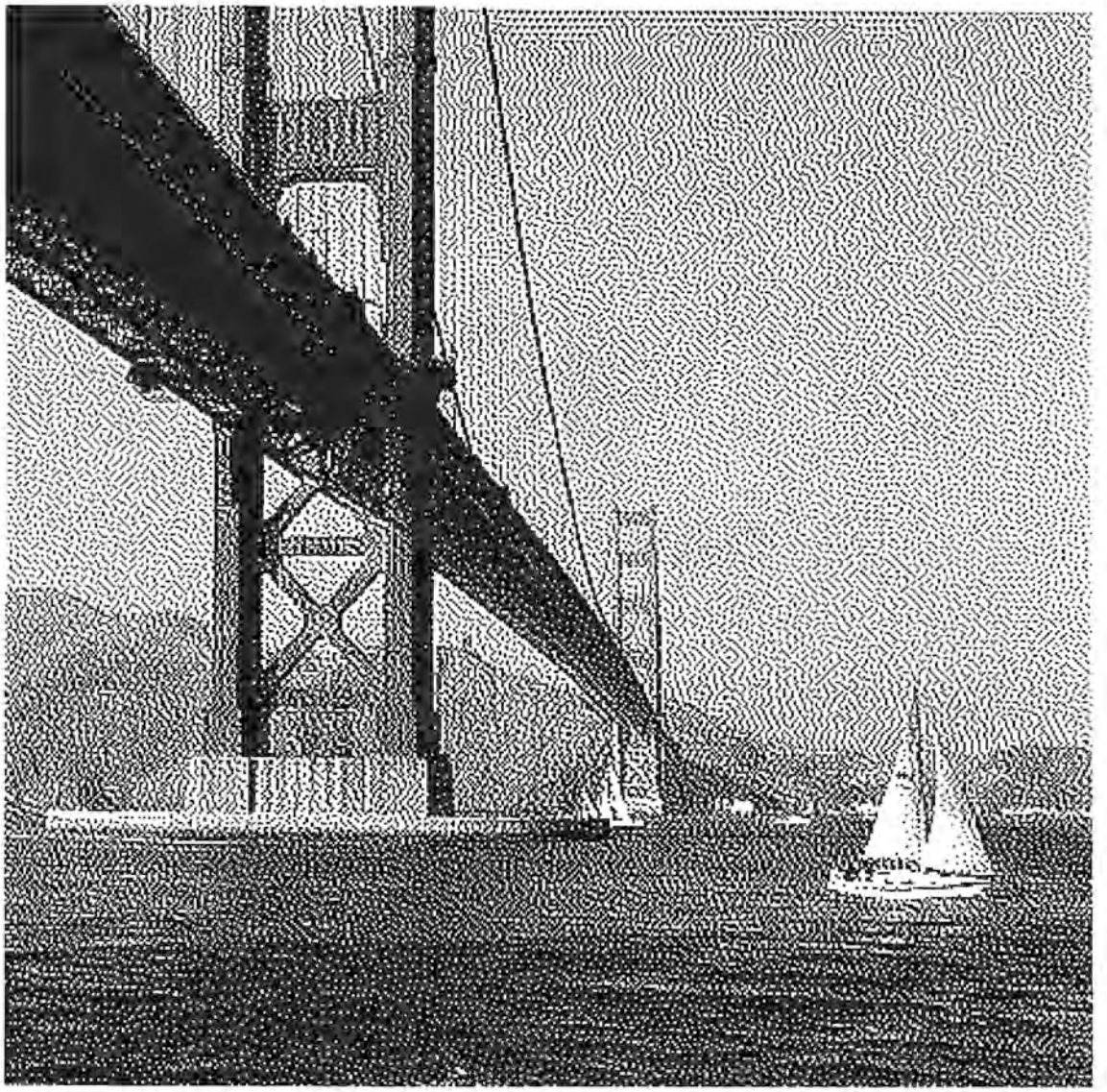

Figure 30. Image #5. Jarvis, Judice, & Ninke Filter. (75 dpi)

U = -49161.00

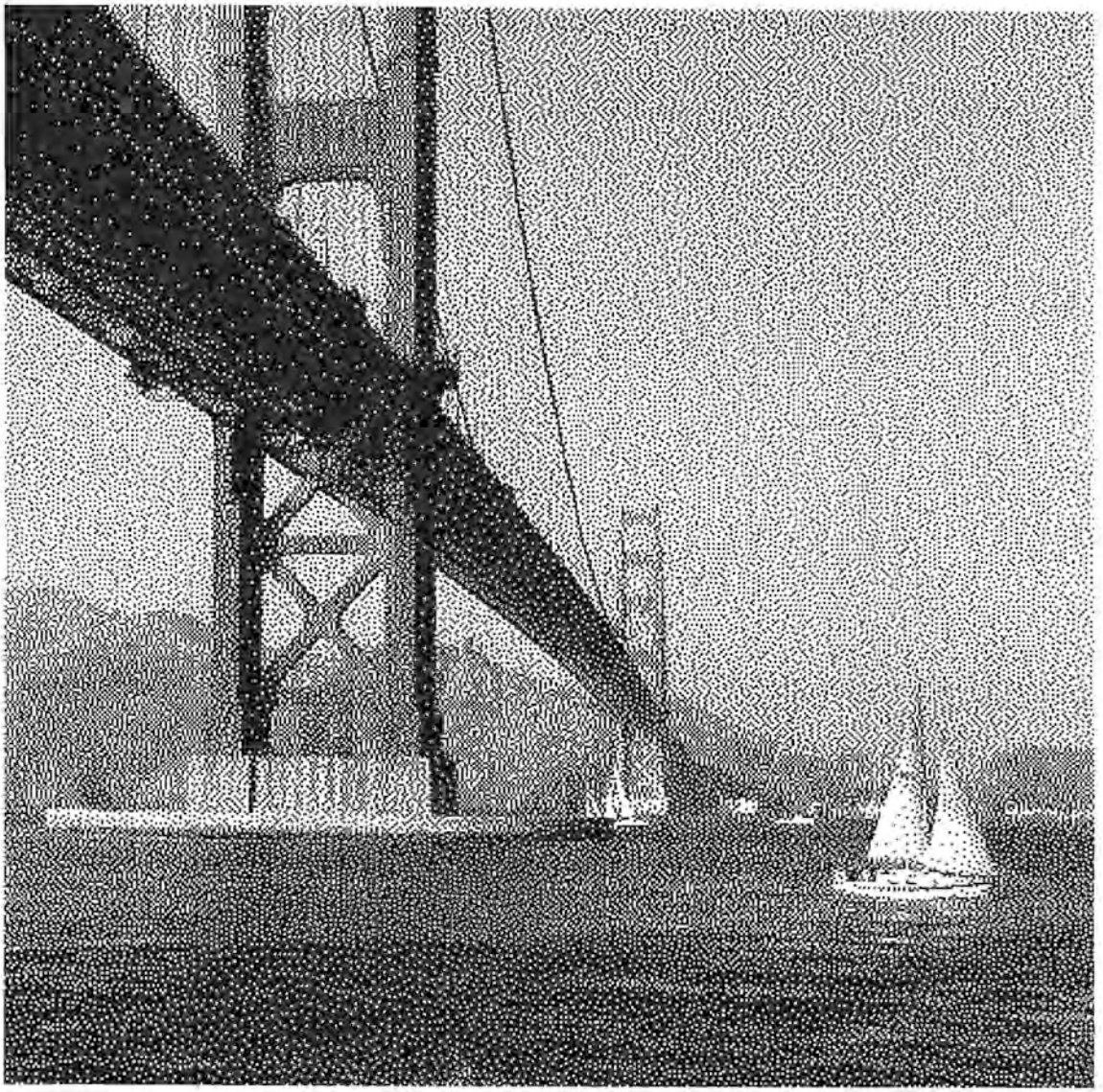

Figure 31. Image #5. 50-Percent Random Floyd-Steinberg Filter. (75 dpi)

U = -49828.81

APPENDIX D

Software Source Code

```
/*   Energy.c

This program calculates the 'Energy Measure' of binary
halftones, using the algorithm published by Geist, Reynolds,
and Suggs in "A Markovian Framework for Digital Halftoning",
ACM Transactions on Graphics, Vol. 12, No 2, April 1993.

The program also calculates binary halftones using three
methods:

     1.   Bayer Ordered Dither (with an 8x8 dither matrix),
     2.   Jarvis, Judice, and Ninke Minimal Average Error
          (Error Diffusion), &
     3.   Ulichney's 50%-Random-Weighted Serpentine Raster
          version of the Floyd-Steinberg Error Diffusion
          Filter.
*/


#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "bitmap.h"     /* Listing follows Energy.c  */

#define MAXCOLOR 255


/* define image size. */
#define XSIZE 440
#define YSIZE 440

#define RADIUS (5)
#define MAXNEIGHBORS (80)
```

```
/* calculate the horizontal position of pixel i within the
image */

long int x(long int i)
{
     return (i % XSIZE);
}


/* calculate the vertical position of pixel i within the
image */

long int y(long int i)
{
     return ((i / XSIZE) + 1);
}


/* return the distance between two pixels */

double distance(long int i, long int j)
{
     long int dx, dy;

     dx = x(i)-x(j);
     dy = y(i)-y(j);
     return(sqrt((double)dx*dx+dy*dy));
}



/* i and j are neighbors if they are within distance RADIUS
of each other,
          and are not the same pixel. */

int neighbor(long int i, long int j)
{
     if ((distance(i, j) <= RADIUS) && (i != j))
          return (1);
     else
          return (0);
}
```

```
/* return the value of the binary halftone at pixel i. (00
or FF) */

int w(long int i, struct bm256 *halftone)
{
     return (halftone->array[i] / 255);
}



/* return the original value at pixel i as a floating point
     number, scaled to 0-1. */

double V(long int i, struct bm256 *orig)
{
     return ((double)orig->array[i] / 255.0);
}


/* returns the average pixel value for all neighbors of i
     (including i) */

unsigned char char_AVE(long int i, struct bm256 *orig)
{
     long int j, pixels, count, height, first, last;
     long int sum, num;

     pixels = (orig->xs+1)*(orig->ys+1);

     /* start sum to include the value at pixel i. This is
     necessary, since i is not defined as a neighbor of i.
     */

     sum = orig->array[i];
     num = 1;

     height = (XSIZE * RADIUS);
     first = max(0, i-height-1);
     last = min(pixels, i+height+1);

     /* The purpose of count, first, and last is simply to
     speed calculations.
```

```
      The algorithm states to sum for all i,j. But for a
      given pixel i, there can be no more than 80 neighbors j
      (given the radius 5).

      If i and j are not neighbors, then the energy cannot
      change.
      So using count, MAXNEIGHBORS, and first, and last are
      just some simple optimizations to speed things along.*/

      count = 0;
      for (j=first; (j < last)&&(count < MAXNEIGHBORS); j++)
           {
           if (neighbor(i, j))
                {
                /* found a neighbor, increment count */
                count++;

                /* add pixel value to sum */
                sum += orig->array[j];
                /* count the pixel */
                num++;
                }
           }
      return(sum/num);      /* return the average value */
}



/* The avgs matrix is stored as integer values.   This
      returns the average value as a floating point number,
      scaled to 0-1. */

double double_AVE(long int i, struct bm256 *avgs)
{
      return(avgs->array[i]/255.0);
}
```

```
/* calculate the matrix of average values. */
void calc_avgs(struct bm256 *orig, struct bm256 *avgs)
{
     long int i, pixels;
     pixels = (orig->xs+1) * (orig->ys+1);

     /* for all pixels */
     for (i = 0; i < pixels; i++)
          {

          /* calculate the average value of all neighbors to
               i (including i) */
          avgs->array[i] = char_AVE(i, orig);
          };
}



/* rho is the coefficient obtained by inverting the
idealized power spectrum for halftones.  */

double rho(double k, double pf)
{
     double b, top, spikeup, spikedown, kp, r;
     double pi = 3.14159265;

     b = 0.8 * pf;
     top  = (0.4 * (sqrt(2.0) * pf + 1));
     spikeup = 1.05 * pf;
     spikedown = 0.95 * pf;
     kp = k*pi;

     r = (sin(kp*spikeup) - sin(kp*spikedown)) / (4*kp);
     r += (cos(kp*top) - cos(kp*b)) / ((top-b)*kp*kp);

     return (r);
}
```

```
/* given two pixels i,j, and the matrix of average values,
function find_T calculates the value of the T matrix at i,j.

Note in this implementation, the T matrix is not stored, but
calculated as needed.

Since the T matrix is an array of floating point values,
indexed by the number of pixels times 40 (half the
neighbors), the memory requirements for storage can be quite
large.  This method is a compromise of speed for efficiency.
  */

double find_T(long int i, long int j, struct bm256 *avgs)
{
     const double A = 0.15;
     const double B = 0.03;
     double k, mean, pfreq, T;

     if (!neighbor (i, j))
          return (0.0);
     else
          {
          k = distance (i, j);

          mean=0.5*(double_AVE(i,avgs)+double_AVE(j, avgs));
          if (mean <= 0.5)
               pfreq = sqrt (mean);
          else
               pfreq = sqrt (1 - mean);
          }

     T = (A * rho(k, pfreq) - B / (k * k));
     return(T);
}


/* Function total_energy calculates and returns the energy
for the supplied gray-scale image and its binary halftone.

Note that the matrix of averages for all pixels must be
calculated before the call */
```

```
double total_energy(struct bm256 *orig, struct bm256
     *halftone, struct bm256 *avgs)
{
     long int i, j, pixels, count, height, first, last;
     long int xmax, ymax;
     double energy;

     height = (XSIZE * RADIUS);

     energy = 0.0;

     xmax = min(orig->xs, halftone->xs);
     ymax = min(orig->ys, halftone->ys);

     /* total number of pixels in the image */
     pixels = (xmax+1)*(ymax+1);


     /*   First pass.
     This calculates the second term of the energy value:

               -SUM_i ((2V_i - 1)(2w_i - 1))

     This term compares the similarity of each pixel in the
     halftone to its original gray scale value. */

     for (i = 0; i < pixels; i++)
          {
          energy -= (2* V(i, orig) - 1) *
                         (2* w(i, halftone) - 1);
          }


     /*   Second pass.
     This calculates the first term of the energy value:

      -(1/2)SUM_i (SUM_j ( T_i,j   * (2w_i - 1)(2w_j - 1) ))

     The purpose of this term is to measure the 'blue-noise'
     quality of the region around pixel i, and the overall
     intensity of the region. */
```

```
    for (i = 0; i < pixels; i++)
        {

        /* The purpose of count, first, and last is simply
        to speed calculations.

        The algorithm states to sum for all i,j.      But
        for a given pixel i, there can be no more than 80
        neighbors j (given the radius 5).

        If i and j are not neighbors, then the energy
        cannot change.
        So using count, MAXNEIGHBORS, and first, and last
        are just some simple optimizations to speed things
        along.     */

        count = 0;
        first = max(0, i-height-1);
        last = min(pixels, i+height+1);

        for (j=first;(j<last)&&(count < MAXNEIGHBORS);j++)
            {
            if (neighbor(i, j))
                {
                /* found a neighbor, increment count */
                count++;

    /* the following is an optimized expression for:

            -(1/2) T_{i,j} * (2w_i - 1)(2w_j - 1)              */

                if (w(i, halftone) == w(j, halftone))
                    energy -= .5 * find_T(i,j, avgs);
                else
                    energy += .5 * find_T(i,j, avgs);
                }
            }
        }
    return (energy);     /* done.  return energy value */
}
```

```
void bayer8(struct bm256 *source, struct bm256 *dest)
{
     int x, y, i, j, n;
     unsigned char c;
     int xmax, ymax;

     /* matrix of weights for optimal ordered dither, as
     determined by Bayer */

     int D[8][8] = { {  0, 32,  8, 40,  2, 34, 10, 42 },
                     { 48, 16, 56, 24, 60, 18, 58, 26 },
                     { 12, 44,  4, 36, 14, 46,  6, 38 },
                     { 60, 28, 52, 20, 62, 30, 54, 22 },
                     {  3, 35, 11, 43,  1, 33,  9, 41 },
                     { 51, 19, 59, 27, 49, 17, 57, 25 },
                     { 15, 47,  7, 39, 13, 45,  5, 37 },
                     { 63, 31, 55, 23, 61, 29, 53, 21 } };
     n = 8;

     xmax = min(source->xs, dest->xs);
     ymax = min(source->ys, dest->ys);
     copybitmap (source, dest);

     /* process each pixel in the bitmap.
          Note order is not important */

     for (x = 0; x <= xmax; x++)
          for (y = 0; y <= ymax; y++)
               {
               /* determine position within the repeating
               matrix D[][] */

               i = (x % n);    /* i = horizontal offset */
               j = (y % n);    /* j = vertical offset */

               /* get original pixel color */
               c = findcolor(x, y, source);

               /* First scale the matrix value up from 0-63
               to 0-255.

               The 8x8 dither matrix can only produce 64
               intensity patterns.
```

```
                Next, use the value as a threshold.
                Then store the resulting 00 or FF.  */

                storecolor (x,y,dest,
                                (c > (256/n/n)*D[i][j])*255);
                };
}


void jarvis(struct bm256 *source, struct bm256 *dest)
{
     int x, y;
     unsigned char c, newc;
     char error;
     int width, height;
     width     = min(source->xs, dest->xs);
     height = min(source->ys, dest->ys);
     copybitmap (source, dest);


     /* process scan lines from top to bottom */
     for (y = 0; y <= height; y++)

          /* process each scan line from left to right */
          for (x = 0; x <= width; x++)
               {
               /* get pixel value from original image */
               c = findcolor(x, y, dest);

               /* Use a simple threshold of 127.  Determine
               the error value */

               if (c > 127)
                    {
                    newc = 255;
                    error = c - 255;
                    }
               else
                    {
                    newc = 0;
                    error = c;
                    }
```

```
                /* store the binary (00 or FF) value dictated
                by the threshold test */

                storecolor (x, y, dest, newc);

                /* distribute error value to neighboring
                pixels.

                The Jarvis, Judice, and Ninke filter may be
                illustrated as:

                      X 7 5

                  3 5 7 5 3

                  1 3 5 3 1                              */

                increasecolor (x+1,  y, dest, (error) *7/48);
                increasecolor (x+2,  y, dest, (error) *5/48);
                increasecolor (x-2,y+1, dest, (error) *3/48);
                increasecolor (x-1,y+1, dest, (error) *5/48);
                increasecolor (x  ,y+1, dest, (error) *7/48);
                increasecolor (x+1,y+1, dest, (error) *5/48);
                increasecolor (x+2,y+1, dest, (error) *3/48);
                increasecolor (x-2,y+2, dest, (error) *1/48);
                increasecolor (x-1,y+2, dest, (error) *3/48);
                increasecolor (x  ,y+2, dest, (error) *5/48);
                increasecolor (x+1,y+2, dest, (error) *3/48);
                increasecolor (x+2,y+2, dest, (error) *1/48);
                };
}


/*function random(x) returns an integer between 0 and x-1.*/
int random(int range)
{
     float f;

     f = rand()/32768.0;
     return((int)(f*range));
}
```

```
void rs50_floyd_steinberg(struct bm256 *orig,
                          struct bm256 *halftone)
{
     int x, y, xx;
     unsigned char c, newc;
     char error;
     int width, height;
     int rlarge, rsmall;
     width     = min(orig->xs, halftone->xs);
     height = min(orig->ys, halftone->ys);
     copybitmap (orig, halftone);

     /* process scan lines from top to bottom */
     for (y = 0; y <= height; y++)

          /* process pixels across the current scanline */
          for (xx = 0; xx <= width; xx++)
               {
               /* test for raster direction, to use
               'serpentine raster' */

               if (y % 2)
                    x = xx;   /* processing left to right */
               else
                    x = width-xx; /* process right to left*/

               /* get current color */
               c = findcolor(x, y, halftone);

               /* apply threshold of 127, calculate error */
               if (c > 127)
                    {
                    newc = 255;
                    error = c - 255;
                    }
               else
                    {
                    newc = 0;
                    error = c;
                    }

               /* store color dictated by threshold test */
               storecolor (x, y, halftone, newc);
```

```
/* int from -5 to +5  */
rlarge = random(11) - 5;

/* int from -1 to +1  */
rsmall = random(3) - 1;


/* distribute error value using randomized
weights, to neighboring pixels.

The Floyd-Steinberg filter is normally
expressed as:

      X 7

    3 5 1

Here, the weights are scaled by two to
facilitate the 50% randomly perturbed
weights, while still using integer
arithmetic.  The filter is therefore:

      X   14

 6   10    2                  */

if (y % 2)  /* test for raster direction */
  {
  increasecolor (x+1,  y, halftone,
                (int)(error*(14+rlarge)/32));
  increasecolor (x-1,y+1, halftone,
                (int)(error*( 6+rsmall)/32));
  increasecolor (  x,y+1, halftone,
                (int)(error*(10-rlarge)/32));
  increasecolor (x+1,y+1, halftone,
                (int)(error*( 2-rsmall)/32));
  }
else
  {
  increasecolor (x-1, y  , halftone,
                (int)(error*(14+rlarge)/32));
  increasecolor (x+1, y+1, halftone,
                (int)(error*( 6+rsmall)/32));
```

```
                    increasecolor (x   , y+1, halftone,
                                   (int)(error*(10-rlarge)/32));
                    increasecolor (x-1, y+1, halftone,
                                   (int)(error*( 2-rsmall)/32));
                    }
                  };
}


int main(void)
{
     struct bm256 bitmap1, bitmap2, avgs;
     unsigned long arraysize;
     double U;

     char filename[40] = "";
     char filename2[40] = "";

     arraysize = (unsigned long)XSIZE*(unsigned
          long)YSIZE*sizeof(unsigned char);

     /* allocate memory to store the original gray-scale
     image */

     if ((bitmap1.array =
          (unsigned char *)malloc(arraysize)) == NULL)
          {
          perror("bitmap1[] memory allocation failed.");
          return(0);              /* exit on failure */
          }
     bitmap1.xs = XSIZE-1;
     bitmap1.ys = YSIZE-1;

     /* allocate memory to store the binary halftone created
     from bitmap1

     NOTE: For simplicity, the binary halftone is still
     stored as an 8-bit per pixel array, but having only
     values 00 and FF */

     if ((bitmap2.array =
          (unsigned char *)malloc(arraysize)) == NULL)
          {
```

```
     perror("bitmap2[] memory allocation failed.");
     return (0);          /* exit on failure */
     }
bitmap2.xs = XSIZE-1;
bitmap2.ys = YSIZE-1;

/* allocate memory for the avgs[] array, used in energy
calculations */

if ((avgs.array = (unsigned char *)malloc(arraysize))
     == NULL)
     {
     perror("avgs[] memory allocation failed.");
     return (0);          /* exit on failure */
     }

/* avgs[] is same size as the bitmap being processed */
avgs.xs = XSIZE-1;
avgs.ys = YSIZE-1;

/* ask for the filename of the gray-scale image */
printf("Enter byte array file name:");
scanf("%s",&filename);

/* make a filename for recording the avgs[] array for
the image */
strcpy(filename2, filename);
strcat(filename2, ".avgs");


/* read the gray-scale image file */
readbytearray (filename, &bitmap1, XSIZE, YSIZE);

/* if the avgs[] array has not been computed yet, do
so, then save it so it doesn't have to be recalculated
for every halftone of this image */

if (readbytearray(filename2, &avgs, XSIZE, YSIZE))
      {
      calc_avgs(&bitmap1, &avgs);
      writebytearray(filename2, &avgs, XSIZE, YSIZE);
      }
```

```
    /* create the Bayer Dither halftone for the image */
    bayer8(&bitmap1, &bitmap2);
    /* store the binary halftone */
    writebytearray("bayer8", &bitmap2, XSIZE, YSIZE);
    /* calculate the energy for the image */
    U = total_energy(&bitmap1, &bitmap2, &avgs);
    /* print the final energy value */
    printf(out, "\nThe total Energy U = %16.4lf\n\n", U);


    /* create the Jarvis, Judice, & Ninke halftone for the
    image */
    jarvis(&bitmap1, &bitmap2);
    /* store the binary halftone */
    writebytearray("jarvis", &bitmap2, XSIZE, YSIZE);
    /* calculate the energy for the image */
    U = total_energy(&bitmap1, &bitmap2, &avgs);
    /* print the final energy value */
    printf(out, "\nThe total Energy U = %16.4lf\n\n", U);

    /* create the 50%-random-weighted Floyd-Steinberg
    halftone for the image */
    rs50_floyd_steinberg(&bitmap1, &bitmap2);
    /* store the binary halftone */
    writebytearray("rs_floyd", &bitmap2, XSIZE, YSIZE);
    /* calculate the energy for the image */
    U = total_energy(&bitmap1, &bitmap2, &avgs);

    /* print the final energy value */
    printf(out, "\nThe total Energy U = %16.4lf\n\n", U);

    /* free memory allocated for bitmaps */
    free(bitmap1.array);
    free(bitmap2.array);
    free(avgs.array);

    return (0);
}

/* end of file "energy.c" */
```

```
/*   Bitmap.h

This is a set of routines for manipulating gray-scale
bitmaps as arrays of character values.      */

#ifndef _BITMAP_

#define _BITMAP_
#include <stdlib.h>
#include <stdio.h>

#define MAXXSIZE 512

#define max(value1,value2) ((value1>value2) ? value1:value2)
#define min(value1,value2) ((value2>value1) ? value1:value2)


/* store bitmaps as structures, containing the width,
height, and a pointer to an array. Memory must be allocated
for the array, since it is not static.       */

struct bm256 {
     int xs, ys;
     unsigned char *array;
};


/* store the pixel value c in the bitmap, at x,y */
void storecolor (int x, int y, struct bm256 *bitmap,
     unsigned char c)
{
     int xmax, ymax;
     unsigned long offset;

     xmax = bitmap->xs;
     ymax = bitmap->ys;
     if ((x <= xmax) && (y <= ymax) && (x >= 0) && (y >= 0))
          {
          offset = (unsigned long)x + (unsigned long)y *
               (unsigned long)(xmax+1);
          bitmap->array[offset] = c;
          }
}
```

```
/* increase (or decrease) the value of pixel x,y by the
amount dcolor */

void increasecolor (int x, int y, struct bm256 *bitmap, char
     dcolor)
{
     int xmax, ymax, c;
     unsigned long offset;

     xmax = bitmap->xs;
     ymax = bitmap->ys;
     if ((x <= xmax) && (y <= ymax) && (x >= 0) && (y >= 0))
          {
          offset = (unsigned long)x + (unsigned long)y *
               (unsigned long)(xmax+1);

          c = bitmap->array[offset];
          bitmap->array[offset]=max(0, min(255,(c+dcolor)));
          }
}


/* retrieve the value of the pixel x,y in bitmap */

unsigned char findcolor (int x, int y, struct bm256 *bitmap)
{
     int xmax, ymax;
     unsigned long offset;

     xmax = bitmap->xs;
     ymax = bitmap->ys;
     if ((x > xmax) || (y > ymax))
          {
          perror ("coordinate out of bitmap's range.\n");
          return (0);
          }
     else
          {
          offset = (unsigned long)x + (unsigned long)y *
               (unsigned long)(xmax+1);
          return (bitmap->array[offset]);
          }
}
```

```
/* Copy one bitmap to another */
void copybitmap(struct bm256 *source, struct bm256 *dest)
{
     int x, y;
     unsigned char c;
     int width, height;
     width     = min(source->xs, dest->xs);
     height = min(source->ys, dest->ys);

     for (x = 0; x <= width; x++)
          for (y = 0; y <= height; y++)
               {
               c = findcolor(x, y, source);
               storecolor (x, y, dest, c);
               };
}


/* Read a raw data file from disk into a bitmap array */
int readbytearray(char *filename, struct bm256 *bitmap, int
     xs, int ys)
{
     int x, y, xmax, ymax;
     unsigned char buf[MAXXSIZE];
     FILE *fp;
     xmax = bitmap->xs;
     ymax = bitmap->ys;

     if ((fp = fopen(filename, "rb")) == NULL)
          return (-1);
     for (y = 0; y <= min(ys-1, ymax); y++)
          {
          if (fread(buf, 1, xs, fp) <= 0)
               {
               perror("readbytearray: bad read");
               return (-1);
               }
          for (x = 0; x <= min(xmax, xs-1); x++)
               storecolor(x, y, bitmap, buf[x]);
          }
     fclose(fp);
     return (0);
}
```

```
/* write a raw data file to disk from a bitmap array */

int writebytearray(char *filename, struct bm256 *bitmap, int
     xs, int ys)
{
     int x, y, xmax, ymax;
     unsigned char buf[MAXXSIZE];
     FILE *fp;

     xmax = bitmap->xs;
     ymax = bitmap->ys;

     if ((fp = fopen(filename, "wb")) == NULL)
          return (-1);
     for (y = 0; y <= min(ys-1, ymax); y++)
          {
          for (x = 0; x <= min(xmax, xs-1); x++)
               buf[x]=(unsigned char)findcolor(x,y, bitmap);
          if (fwrite(buf, 1, xs, fp) <= 0)
               {
               perror("writebytearray: bad write");
               return (-1);
               }
          }
     fclose(fp);
     return (0);
}

#endif

/* end of file  "bitmap.h" */
```

APPENDIX E

The Energy Measure

Original gray-scale image:

An n x n array V of real numbers, $V_{i,j} \in [0,1]$

Binary halftone image:

An n x n array ω of binary integers, $\omega_{i,j} \in \{0,1\}$

1. Individual binary pixel values should be strongly correlated to the gray-scale of the underlying pixels.

$$U(\{i\}) = (1 - 2\omega_i)(2V_i - 1)$$

2. In any small neighborhood, the binary pixels should create a pattern which accurately represents the average gray-scale intensity of that neighborhood. Also, this pattern should have a minimum of low-frequency noise.

$$U(\{i,j\}) = (1 - 2\omega_i)(1 - 2\omega_j)\ T(i,j)$$

Total energy for a given binary image is therefore:

$$(-1/2)\ \Sigma_i\Sigma_j T_{i,j}(2\omega_i - 1)(2\omega_j - 1) - \Sigma_i(2\omega_i - 1)(2V_i - 1)$$

Calculation of the T-matrix is based on the desired spectral characteristics determined by Ulichney. This is the algorithm given by Geist, Reynolds, and Suggs.

```
compute_T_matrix()
  {
  #define A (0.15)
  #define B (0.03)

  for (I = 1; I <= PIXELS; I = I+1)
     AVE_i = average gray-scale intensity for all pixels in
               NEIGHBORS(I);

  for (I = 1; I <= PIXELS; I = I+1)
     {
     for (J ∈ NEIGHBORS(I))
        {
        MEAN = (AVE_i + AVE_j)/2;
        if (MEAN <= 0.5) then
              PFREQ = square_root(MEAN)
           else
              PFREQ = square_root(1 - MEAN);

         K = distance(I, J);
         T_i,j = A * rho(K,PFREQ) - B/K^2;
         }
     }
  }

rho(K, PFREQ)
  {
  base       = 0.8 * PFREQ;
  top        = 0.4 * (square_root(2) * PFREQ + 1);
  spike_up   = 1.05 * PFREQ;
  spike_down = 0.95 * PFREQ;

  kp = K * π;
  rho = (sin(kp*spike_up) - sin(kp*spike_down)) / (4*kp);
      + (cos(kp*top) - cos(kp*b)) / ((top-b)*kp*kp);
  }
```

APPENDIX F

Error Diffusion Filters

The standard error diffusion technique considers each pixel in the image moving from top to bottom, processing each line left to right. Each pixel is compared to a threshold. The "error," or difference between the desired intensity and the halftone intensity chosen, is distributed in a weighted fashion to adjacent pixels. Consider the Floyd-Steinberg filter (a matrix of weights) shown below.

Floyd-Steinberg: • 7

3 5 1

The dot represents the pixel being halftoned at a particular instant, and the numbers indicate the relative amount of error to distribute to neighboring pixels. Since the four numbers sum to sixteen in this filter, each weight must be divided by sixteen before multiplying by the error, thus distributing exactly 100% of the error each time. For example, the pixel immediately to the right will always receive $7/16^{ths}$ of the error, while the pixel straight below receives $5/16^{ths}$ of it. The other error diffusion filters work exactly the same way, but differ in the placement and amount of the error distribution.

Table VI. Error Diffusion Filters

Floyd-Steinberg:

$$\begin{array}{ccc} \bullet & 7 & \\ 3 & 5 & 1 \end{array}$$

Burkes

$$\begin{array}{ccccc} & & \bullet & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \end{array}$$

Jarvis, Judice, & Ninke

$$\begin{array}{ccccc} & & \bullet & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{array}$$

Stucki

$$\begin{array}{ccccc} & & \bullet & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{array}$$