Global Optimization of Dither Arrays

by

Jeffrey Newbern

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

© Jeffrey Newbern, MCMXCVI. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly
paper and electronic copies of this thesis document in whole or in part, and to grant
others the right to do so.

Author

Department of Electrical Engineering and Computer Science

May 20, 1996

Certified by

V. Michael Bove
Associate Professor

Accepted by

F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Students
Global Optimization of Dither Arrays

by

Jeffrey Newbern

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 1996, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Genetic search techniques are applied to the task of generating threshold arrays for
digital halftoning. A hybrid genetic algorithm is developed to search the space of
threshold arrays for those which exhibit desirable visual characteristics. The task re-
quires a specialized mutation strategy and efficient routines for evaluating candidate
arrays. The resulting algorithm produces high-quality blue noise masks with fewer
low-frequency artifacts than other methods. The genetic technique provides a greater
degree of flexibility and control over the characteristics of the threshold array than
heuristic-based algorithms. It requires a significantly greater computational invest-
ment than other techniques for small arrays but its efficiency is comparable to those
techniques for large arrays.

Thesis Supervisor: V. Michael Bove
Title: Associate Professor
Acknowledgments

To Prof. Bove, for his technical advice. To Bill Light and Digital Equipment Corp., for their support. To my parents, whose financial backing made this thesis possible. And to my roommate Raj, who has been drumming continuously for four hours now.
Contents

1 Introduction

1.1 Digital Halftoning .................................................. 9
1.2 Halftoning with Threshold Arrays ............................. 10
1.3 Desirable Properties of Halftoned Images ................. 12
   1.3.1 Tone Scale Reproduction .................................. 12
   1.3.2 Low-Frequency Artifacts .................................. 14
   1.3.3 Textures .................................................... 14
   1.3.4 Blue Noise ................................................ 14
1.4 Generating Blue Noise Masks ................................. 15

2 Genetic Search Techniques ............................. 17

2.1 Breeding and Population Structure ....................... 17
   2.1.1 Population Size .......................................... 18
   2.1.2 Population Structure .................................... 18
   2.1.3 Selection Strategy ....................................... 19
   2.1.4 Replacement Strategy ................................... 19
2.2 Fitness Evaluation ........................................... 20
2.3 Mutation ....................................................... 21
   2.3.1 Crossover ................................................... 21
   2.3.2 Point Mutation ............................................ 21
   2.3.3 Hybrid Mutators ........................................ 22
3  A Genetic Method for Generation of Threshold Arrays

3.1  Threshold Array Representation ............................................ 23
3.2  Algorithm Structure .............................................................. 24
   3.2.1  Breeding Population ...................................................... 24
   3.2.2  Selection ........................................................................ 25
3.3  Fitness Evaluation .................................................................... 25
   3.3.1  Repulsion ........................................................................ 26
   3.3.2  Smoothness ...................................................................... 29
   3.3.3  Partitioning ..................................................................... 30
3.4  Mutation Strategy ..................................................................... 35
   3.4.1  Order-Based Crossover ..................................................... 35
   3.4.2  Point Swaps ..................................................................... 35
   3.4.3  Point Movement ............................................................... 36
   3.4.4  Operator Reweighting ....................................................... 37
3.5  Important Optimizations .......................................................... 38
   3.5.1  Memoization of Fitness Calculations .................................. 38
   3.5.2  Hints for Mutators ........................................................... 38
   3.5.3  Seeding and Staged Computation ...................................... 42

4  Results ...................................................................................... 43

4.1  Threshold Arrays ................................................................... 43
4.2  Spectral Properties ................................................................. 46
4.3  Evolution of the Population Over Time .................................... 46
4.4  Running Time ......................................................................... 48
4.5  Effects of Fitness Function Components ................................... 50
4.6  Mutation Operator Fitness ....................................................... 50

5  Conclusions .............................................................................. 57
## List of Figures

1-1 Three examples of halftoning by threshold array .......................... 11
1-2 Use of gamma correction to compensate for dot gain .................... 13
1-3 A typical blue noise spectrum ................................................. 15

3-1 Comparison of optimizer performance for five population structures 25
3-2 Repulsion curve for $32 \times 32$ recursive tessellation array .......... 26
3-3 Weighting curve applied to normalized repulsion values ............... 27
3-4 Illustration of pixel categorization at the 50% partitioning level ... 31
3-5 Use of staged computations to generate large arrays .................. 42

4-1 Comparison of ramps halftoned using GA vs standard techniques 44
4-2 Comparison of halftoned ‘Lena’ using GA vs standard techniques 45
4-3 Comparison of spectral properties of blue noise masks ................ 47
4-4 Fitness component curves during a typical run .......................... 48
4-5 Evolution of threshold arrays over time ................................ 49
4-6 Comparison of running times for array generation ...................... 50
4-7 $128 \times 128$ threshold array generated using staged computation ... 51
4-8 Effect of optimizing all components of the fitness function .......... 52
4-9 Effect of repulsion component of fitness function .................... 53
4-10 Effect of smoothness component of fitness function .................. 54
4-11 Mutator fitness curves during a typical run ............................ 55
Chapter 1

Introduction

1.1 Digital Halftoning

Digital halftoning is the process by which a continuous-tone image is approximated using a small set of discrete tones. Digital halftoning is necessary for the reproduction of gray-scale images on devices with limited tone resolution, such as laser printers and binary CRTs. This thesis deals with the case of binary output, but the results are easily generalized to halftoning on devices with a greater number of output tones.

One important and widely-used halftoning technique determines the pattern of pixels in the output image by comparing each pixel in the input image with a fixed array of thresholds tiled across the image. This involves little computation at the time each image is printed. Because the cost of generating a threshold array can be amortized over hundreds of thousands of printers, it is economical to incur a large computational cost to generate it. When using threshold arrays for digital halftoning, the challenge is to compute arrays which generate output patterns with desirable visual characteristics at every gray level. Currently, such arrays are computed using heuristic-based algorithms, such as Robert Ulichney's void-and-cluster algorithm[10], which work well but are not flexible enough to be tuned to specific applications and printing technologies. It is the goal of this thesis to present a flexible method for computing such threshold arrays.

Many threshold arrays used today are generated using a method of recursive
Recursive tessellation arrays are very easy to generate, but they have very distracting periodic structures in the halftone patterns for some gray levels.

A popular alternative to halftoning with threshold arrays is the error diffusion process[4]. Error diffusion is an adaptive process which requires more computation during halftoning than does the thresholding technique. Error diffusion is attractive because it is capable of producing very smooth gray fields and sharp edges. Unfortunately, the adaptive mechanism of error diffusion can fail to generate acceptable halftone patterns for some gray levels. These defects can be overcome by introducing random fluctuations in the error diffusion filter, but other failures are more difficult to address. Error diffusion is not normally used in laser printers because it interacts poorly with the non-ideal pixel shape required by that technology, although progress has been made in overcoming this limitation[6].

1.2 Halftoning with Threshold Arrays

Halftoning with threshold arrays consists of a single point comparison for each pixel in the output image. An $M \times N$ threshold array is tiled across the output page, and each pixel is colored white if the requested output color is greater than the threshold value and black otherwise. Figure 1-1 illustrates the use of a threshold array.

The thresholding point process provides several advantages over halftoning techniques based on neighborhood processes. Halftoning with threshold arrays is computationally efficient, requiring only a single comparison per output pixel, and it is easily parallelized. Because the threshold array completely determines the halftone pattern for each graylevel, threshold arrays offer a measure of control which is not offered by adaptive halftoning processes such as error diffusion. For this reason, it is much easier to ensure that a threshold halftone will not suffer a pathological failure than to prevent such failures from error diffusion.
a) 50% gray

\[
\begin{array}{cccc}
0.5 & 0.5 & 0.5 & 0.5 \\
0.5 & 0.5 & 0.5 & 0.5 \\
0.5 & 0.5 & 0.5 & 0.5 \\
0.5 & 0.5 & 0.5 & 0.5 \\
\end{array}
\]

\[\times \quad \frac{1}{16} \]

\[
\begin{array}{cccc}
0 & 8 & 3 & 11 \\
15 & 4 & 12 & 7 \\
2 & 10 & 1 & 9 \\
13 & 6 & 14 & 5 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
\end{array}
\]

Sampled Continuous-Tone Image  
Threshold Array  
Output Image Bitmap  
Printed Image

b) 20% gray (0% is black, 100% is white)

\[
\begin{array}{cccc}
0.2 & 0.2 & 0.2 & 0.2 \\
0.2 & 0.2 & 0.2 & 0.2 \\
0.2 & 0.2 & 0.2 & 0.2 \\
0.2 & 0.2 & 0.2 & 0.2 \\
\end{array}
\]

\[\times \quad \frac{1}{16} \]

\[
\begin{array}{cccc}
0 & 8 & 3 & 11 \\
15 & 4 & 12 & 7 \\
2 & 10 & 1 & 9 \\
13 & 6 & 14 & 5 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

Sampled Continuous-Tone Image  
Threshold Array  
Output Image Bitmap  
Printed Image

c) Varying intensity (0% is black, 100% is white)

\[
\begin{array}{cccc}
0.8 & 0.7 & 0.6 & 0.5 \\
0.7 & 0.6 & 0.5 & 0.4 \\
0.6 & 0.5 & 0.4 & 0.3 \\
0.5 & 0.4 & 0.3 & 0.2 \\
\end{array}
\]

\[\times \quad \frac{1}{16} \]

\[
\begin{array}{cccc}
0 & 8 & 3 & 11 \\
15 & 4 & 12 & 7 \\
2 & 10 & 1 & 9 \\
13 & 6 & 14 & 5 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
\end{array}
\]

Sampled Continuous-Tone Image  
Threshold Array  
Output Image Bitmap  
Printed Image

Figure 1-1: Three examples of halftoning by threshold array
1.3 Desirable Properties of Halftoned Images

We will concern ourselves with three properties of halftoned images which humans use to form a subjective measure of quality. These properties are *tone scale reproduction*, which measures how well a the halftoned image reproduces the lightness and darkness of the original image, *low-frequency artifacts*, which are visually disturbing and tend to detract from the content of the image, and *texture boundaries*, which are important visual cues that suggest edges or changes in depth which do not exist in the original image.

1.3.1 Tone Scale Reproduction

One of the most important features of a threshold array is its tone scale reproduction. A good *n*-element threshold array should provide *n* + 1 evenly spaced gray levels of $0, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}, \frac{n}{n}$. Such an array is said to provide linear tone scale. When the gray levels are not evenly spaced, a non-linear tone scale results which can cause loss of contrast and detail in regions of the image. The halftoning process attempts to render every gray field as a pattern of black and white dots which most closely matches the desired gray level. A threshold array which does not provide enough gray levels at even intervals will be forced to render some gray fields as halftone patterns which do not match the desired lightness. Such an error is known as *quantization error*.

For an ideal output device, the gray level produced by turning on a fraction $g$ of the pixels will be simply $g$. For such a device, a threshold array with a uniform distribution of threshold values will give perfect tone scale reproduction. Real-world devices, however, are far from ideal. In a write-black laser printer, for example, the size of each black spot rendered onto the page must be greater than the area of an ideal square cell at the printer’s resolution. This fact causes dots from the printer to overlap into neighboring cells, decreasing the amount of white paper showing in the image, resulting in a phenomenon known as “dot overlap” or “dot gain”. Dot gain causes these printers to produce darker images than an ideal output device using the same threshold array.
It is possible to partially compensate for non-linear tone scale reproduction (caused by dot-gain or other non-ideal output characteristics) by preceding the thresholding step by a \textit{gamma correction} process which redistributes the tone scale to make it linear after halftoning. Figure 1-2 illustrates the use of gamma correction to compensate for non-ideal output devices.

When a threshold array fails to provide a fine enough spacing of tone levels in a region of the tone scale \textit{contours}, or visible discontinuities in lightness at nearby points which should be reproduced as only slightly differing grays, can result. Large dot gains require increasingly non-linear gamma correction curves, which can induce unsightly contours in the halftoned image.

The effects of dot overlap can be reduced by clustering dots together in the halftone patterns. The resulting \textit{clustered-dot halftone} suffers less dot gain than a \textit{dispersed-dot halftone}, at the expense of lower spatial resolution. In addition, some output devices cannot reliably reproduce a single pixel, necessitating clustered halftones on those devices. The clustered dots introduce a low-frequency component which gives the image a gridded appearance. Clustering in halftone patterns should be avoided whenever possible, but is sometimes necessary to produce acceptable tone scale reproduction on devices which suffer from dot gain or inconsistent reproduction of isolated pixels.

The work in this thesis assumes that gamma correction will compensate for any non-idealities in the output device; the threshold arrays produced are all linear.
1.3.2 Low-Frequency Artifacts

The human visual system acts, to a first approximation, as a low-pass filter. In addition, the visual system incorporates machinery which detects lines, and is more sensitive to horizontal and vertical lines than to diagonal lines[9].

Lines, clusters of dots, holes with no dots in an otherwise uniform field, and low-frequency gradients produced by a threshold array will all form visual artifacts which detract from the content of the halftoned image. Dispersed-dot halftones provide high spatial resolution and thus fewer low-frequency artifacts than clustered-dot halftones, but are more sensitive to dot gain.

1.3.3 Textures

The human visual system uses the boundary between textures as visual cues for edge detection and depth perception. Any deviation from a uniform texture generated by a halftone process will be visually disturbing and should be avoided. Such textures can arise from changes in the orientation of a pattern or from pathological behavior of an adaptive halftone process such as error diffusion.

1.3.4 Blue Noise

Images which are halftoned using blue noise patterns tend to be more visually pleasing than halftones which exhibit low-frequency structure, such as clustered-dot halftones and dispersed-dot halftones with strong periodic artifacts[9]. A blue noise pattern is a pattern which has little energy in the low-frequency portion of its spectrum, has significant energy content at some principal frequency, and has a uniform frequency content above the principal frequency. Figure 1-3 shows the spectral density estimate of a blue noise pattern with principal frequency $f_p = \frac{1}{4}$.

True blue noise patterns are aperiodic and free of low-frequency spectral energy. It is not possible to generate an aperiodic pattern using a threshold array, but the controllability and efficiency of threshold arrays makes their use attractive. Threshold
Figure 1-3: A typical blue noise spectrum

arrays can be used to generate periodic patterns which mimic the frequency distribution of blue noise. Such threshold arrays are called blue noise masks.

1.4 Generating Blue Noise Masks

The goal of this thesis is to provide a practical and flexible method for generating blue noise masks. The best technique previously available for generating such threshold arrays is the void-and-cluster method of Robert Ulichney (Digital Equipment Corporation)[10]. The void-and-cluster method uses a greedy algorithm, which makes optimal choices at each gray level but does not attempt any global optimization. It is based on the iterative application of a heuristic which places dots in the largest void (region with few dots) and removes dots from the largest cluster (region with many dots).

The void-and-cluster method produces high-quality blue noise masks. The void-and-cluster method is efficient for small arrays but its $\Theta((M \times N)^3)$ time bound quickly leads to massive computational requirements for large arrays. Because threshold arrays need only be generated once and can then be deployed in millions of printers and used efficiently for each printed page, it is cost-effective to make such computational investments.
A more serious drawback of the void-and-cluster method is its inflexibility. The heuristic which is used in the algorithm leads to good threshold arrays, but cannot be easily tuned to a particular output device or halftoning application.

The method presented in this thesis requires comparable computational investments and produces blue noise masks of comparable quality to those of the void-and-cluster method, but it provides a flexible framework which allows the threshold array designer to control the patterns produced by the threshold array at each gray level.
Chapter 2

Genetic Search Techniques

Genetic search techniques are a class of optimization techniques inspired by biological evolution, but not subject to the same constraints that govern natural evolutionary processes. Briefly, a genetic algorithm maintains a population of candidate solutions, encoded as bit-strings called chromosomes, and combines and alters them to create new candidate solutions. Each candidate is assigned a numerical score, or fitness value according to a fitness function. During each iteration, the most fit members of the current population are selected according to a selection strategy and bred according to a mutation strategy. Their offspring replace the least fit members, causing the average fitness of the population to rise. In this way a genetic search can begin from an initially unfit population and generate improved solutions over time.

The design of a genetic algorithm entails setting many parameters such as population size and structure, selection strategy, breeding strategy, duplication policies, and mutation strategies, which is itself a difficult optimization problem. Often, the proper tuning of these parameters depends on the skill of the algorithm designer and is based on intuition and familiarity with the behavior of the algorithm over many runs.
2.1 Breeding and Population Structure

A genetic algorithm manipulates a population of candidate solutions by selective breeding, ensuring that the most fit chromosomes in the current population are represented more frequently in the next generation. Over time, the chromosomes in the gene pool will tend to increase in fitness and the variation within the gene pool will decrease as unfit chromosomes are removed from the gene pool and mutations are less likely to be beneficial. When the genetic variation within the population becomes very small, the population is said to have converged. It is the goal of the algorithm designer to ensure that the population converges to an optimal solution, or in some cases a near-optimal solution. Convergence to a local maximum of the fitness function can stall the progress of the algorithm and lead to unacceptable solutions.

2.1.1 Population Size

The genetic algorithm designer must determine the size of the breeding population based on a number of constraints. Larger populations can accommodate a greater variety of chromosomes, tend to be more resistant to prematurely converging on local maxima, and generally lead to better solutions than do smaller populations. Because beneficial mutations take longer to become fixed in larger populations, they evolve more slowly and take longer to converge. Larger populations also place greater demands on memory than do small populations; a severe penalty is incurred if the size of the pool of candidate chromosomes exceeds the available physical memory of the computer hosting the algorithm.

2.1.2 Population Structure

Under some circumstances it is beneficial to structure the breeding population into smaller sub-populations with controlled interbreeding. This technique can slow global convergence by allowing multiple populations to independently explore different peaks of the fitness landscape without the currently superior candidate forcing other solutions out of the gene pool. It can also allow more rapid optimization by separating
chromosomes in different regions of the fitness landscape (which seldom produce fit offspring when bred) into separate populations.

Depending on the nature of the fitness landscape and the mutation strategy, the use of sub-populations may be wasteful; it may be more efficient to concentrate computational resources on a slightly larger single population rather than to divide them among multiple sub-populations. The algorithm designer must determine the proper population structure, normally through experimentation.

2.1.3 Selection Strategy

The rate of convergence of the algorithm is also affected by the parent selection strategy. In order to create new offspring one or two parents must be selected from the breeding population. There are many techniques for performing parent selection, but two of the most common are “Roulette Wheel” selection and “Tournament” selection.

In roulette wheel selection, chromosomes are ranked by fitness and a fixed probability of selection is assigned to each rank within the population. Parents are selected according to the probability associated with their rank in the population. For instance, a uniform roulette wheel for a population of size $N$ would give to each individual a slice of the roulette wheel of size $\frac{1}{N}$. Each selection event consists of selecting a member of the population with probability proportional to its share of the roulette wheel. Often, individuals with higher fitness scores are allotted more of the roulette wheel so that the more fit individuals breed preferentially.

In tournament selection, a random subset (usually of size 2, but sometimes larger) of the population is examined, and the most fit individual from the subset is selected. This strategy is highly preferential to the most fit members of the population, and the effect becomes more pronounced as population size increases. Using larger random subsets in each tournament also increases the preference given to the most fit individuals.
2.1.4 Replacement Strategy

Genetic algorithms are classified as either generational or steady-state algorithms, according to their method for selecting individuals to be replaced each generation. Generational algorithms replace the entire population with new offspring on each iteration, whereas steady-state algorithms replace only a portion of the population with each iteration. Normally a steady-state algorithm replaces the least fit members of its population, but other variations are possible. The algorithm designer must determine what percentage of the population to replace during each generation; this number is known as the selection pressure. If the selection pressure is too high, premature convergence may result; if it is too low, the algorithm will require converge too slowly.

An improvement to generational replacement implements elitism in order to make sure that the most fit candidate in the population is always preserved in the subsequent generation.

It is possible to delay convergence by enforcing a policy which requires each individual in the population to be unique, in order to prevent the population from becoming dominated by replicas of a single highly fit chromosome.

2.2 Fitness Evaluation

The fitness evaluator is the only component of a “pure” genetic algorithm which is specific to the function to be optimized. Its sole purpose is to generate a numerical figure of merit which measures the property to be optimized. The fitness evaluation should be made as efficient as possible, since it must be performed many times during each iteration.

The fitness function can be thought of as a mathematical description of a multi-dimensional fitness landscape with peaks and valleys. The most fit chromosomes reside at the peaks of the fitness landscape, but the high dimensionality of the search space and the ruggedness of the fitness terrain make it difficult to locate the globally highest peak. The fitness function should be designed in such a way that it rewards
small beneficial variations in the candidate solutions. A genetic algorithm searches the solution space by concentrating on solutions around the peaks of the fitness landscape, so sloping regions of the fitness landscape provide valuable information about the location of peaks but abrupt steps in the fitness landscape do not. In addition, when the solutions to the optimization problem are required to satisfy a restrictive set of invariants, not all points of the fitness landscape will correspond to acceptable solutions.

2.3 Mutation

In order to provide the variation in candidate solutions which drives the genetic algorithm, mutation operators take parent chromosomes, which are bit-encoded candidate solutions, and produce one or more new chromosomes which differ from the parents. Standard genetic algorithms use at least crossover mutators and point mutators. In some cases, the chromosomes must satisfy some global invariant, and the mutation operators must be specifically designed to preserve the invariant in their offspring.

2.3.1 Crossover

Crossover mutators combine the bits from one parent with the bits from the other. Uniform crossover selects the parent from which to inherit each locus (the locus is the smallest unit which can be mutated within a chromosome) of the chromosome randomly and independently. Single point crossover inherits some initial segment of the chromosome from one parent and the remainder from the other. One important variant of crossover is order-based crossover which preserves the values of one parent, but reorders them in the order of the values from the other parent[2]. Order-based crossover is useful when chromosomes encode permutations of a fixed set of objects. There are many other variants of crossover mutation.

The crossover operators rely on variation in the gene pool to provide novel candidate chromosomes. In a population which has converged to near-uniformity, crossover operators fail to provide new variation and their performance significantly degrades.
2.3.2 Point Mutation

Standard point-mutators modify each bit of an encoded solution with some fixed probability. The probability of mutation is determined by the algorithm designer to provide enough variation to allow the algorithm to work but not so much that many unfit offspring are produced each generation.

Standard randomized point mutation may not be useful when chromosomes must satisfy an invariant which could be violated by a random mutation. In such cases, special point mutators which maintain the invariant must be used.

2.3.3 Hybrid Mutators

It is often desirable to augment the standard mutation operators with application-specific mutators which take advantage of special knowledge about the optimization problem and the chromosome encoding. Such a hybrid algorithm may combine genetic search techniques with specialized “directed” mutators which make intelligent guesses about promising modifications to the gene pool or mutators which perform local searches of the fitness landscape.

Hybridization requires the algorithm designer to have a detailed understanding of the optimization problem, and works best when the algorithm is instrumented with mechanisms to provide the designer with feedback concerning the efficiency of hybrid mutators.
Chapter 3

A Genetic Method for Generation of Threshold Arrays

Genetic search techniques promise greater flexibility than heuristic-based methods for the generation of high-quality threshold arrays because the fitness landscape can be easily altered to favor or penalize particular threshold properties.

This chapter presents a hybrid genetic algorithm for creating threshold arrays. The algorithm uses hybrid genetic search to breed high-quality threshold arrays from a population of lower quality. The threshold arrays are ranked according to a fitness function which rewards uniform distribution of pixels at each gray level and small deviation from the desired gray output, and it penalizes certain textures such as checkerboards at 50% gray. The threshold arrays are mutated by specialized mutation operators which maintain linear tone scale reproduction and rely on information about the halftoning process to perform directed mutation.

3.1 Threshold Array Representation

Each threshold array is encoded as a 2-dimensional matrix of unsigned 8-bit values. Gray values of 0.0 (black) correspond to threshold values of 0 and gray values of 1.0 (white) correspond to threshold values of 256. Because the highest representable threshold value is 255, a gray value of 1.0 will always be greater than every threshold,
causing every output pixel to be painted white.

The chromosome containing an $M \times N$ threshold matrix consists of $M \times N$ bytes of threshold data and 1198 bytes of auxiliary data. The size of this data structure places a practical limit on the population size because performance degrades severely when the size of the working set exceeds the memory available. The auxiliary data which is kept within each chromosome consists of bookkeeping data, such as the width, height and fitness of the chromosome, as well as data which is used to store (or memoize) the values of complex calculations which are likely to be needed again. The bulk of the auxiliary data consists of 256 floating-point values which memoize partial results of one component of the fitness function. In addition, each chromosome's auxiliary data includes hints used by the mutation operators to modify the chromosome.

3.2 Algorithm Structure

Experiments with various population structures and sizes indicate that the most efficient optimization is obtained with a small number of sub-populations totaling a few hundred individuals and subject to aggressive selection pressure. Convergence problems which might normally accompany such aggressive optimization are avoided by the use of directed mutation operators.

Figure 3-1 shows the results of five runs of the algorithm in varying population configurations. Each run held the computational investment constant at 100 candidates generated and tested per generation; only the size and structure of the population was varied. These experiments show that small populations with high selection pressure vastly outperform other configurations.

3.2.1 Breeding Population

The breeding population is divided into two sub-populations. Breeding occurs mostly between individuals of the same sub-population and only rarely between individuals from different sub-populations. The algorithm maintains 1 populations of 250 individuals.
Figure 3-1: Comparison of optimizer performance for five population structures. Each run tested 250,000 candidate solutions.

This structure was chosen based on experiments which showed that the benefits of a split population are outweighed by the penalty of redundant calculations for this application, particularly with larger threshold array sizes.

### 3.2.2 Selection

Selection is based on a roulette wheel which selects a member of rank $r$ in a population of size $N$ with probability $\frac{10}{55N} \times ([10^{\frac{N-r}{N}}] + 1)$. This selects the most fit individual ten times more often than the least fit.

### 3.3 Fitness Evaluation

The fitness function employed by this genetic algorithm is calculated in three parts, the repulsion factor ($\mathcal{R}\mathcal{E}\mathcal{P}$), smoothness factor ($\mathcal{S}\mathcal{M}\mathcal{O}\mathcal{T}\mathcal{H}$) and partitioning factor ($\mathcal{P}\mathcal{A}\mathcal{R}\mathcal{T}$). The first two reward a particular desirable property of the threshold array; the last penalizes undesirable properties. The fitness function $F$ is the product of these three components:

$$F(c) = 100 \times \mathcal{R}\mathcal{E}\mathcal{P}(c) \times \mathcal{S}\mathcal{M}\mathcal{O}\mathcal{T}\mathcal{H}(c) \times \mathcal{P}\mathcal{A}\mathcal{R}\mathcal{T}(c)$$  \hspace{1cm} (3.1)
3.3.1 Repulsion

The repulsion component of the fitness function measures the degree to which the minority pixels (white pixels in a pattern below 50% gray or black pixels in a pattern above 50% gray) are uniformly distributed at each gray level produced by the threshold array. At each gray level, \( g \), the repulsion routine computes \( r_g \), the average over all minority pixels of the sum of \( e^{-\frac{d^2}{2g}} \), where \( d \) is the shortest straight-line distance between the pixel and each other minority pixel in a \( 19 \times 19 \) cell neighborhood of the halftone pattern around the pixel. A \( 19 \times 19 \) neighborhood is used because it is large enough to yield meaningful results for halftone patterns at \( \frac{1}{256} \) gray, when only one in 256 pixels will be a minority pixel. A measure of the deviation from uniformity is obtained by taking the weighted average of \( r_g \) normalized against the \( r_g \) obtained by using a recursive tessellation array to generate the halftone patterns. Any normalized \( r_g \) greater than 10 is replaced by \( 7.7 + \log(r_g) \) in the calculation of the weighted average, to prevent the optimization from being dominated by large deviations which may be acceptable under some circumstances.

Figure 3-2 shows the normalization curve for a \( 32 \times 32 \) recursive tessellation array.

The normalized \( r_g \) values are weighted to allow greater deviations in the extreme regions of the tone scale, where the small number of minority pixels lead to large
Figure 3-3: Weighting curve applied to normalized repulsion values

deviations from the recursive tessellation values, even for acceptable threshold arrays. The weighting also places less emphasis on the repulsion values in the midtones, where the proper halftone patterns are better determined by other means. Figure 3-3 shows the weighting applied to the normalized $r_g$ values.

The $REP$ component of the fitness function is computed as

$$REP(c) = e^{-(WRMS(c)-1)}$$  \hspace{1cm} (3.2)

where $WRMS(c)$ is the weighted root-mean-square value computed using the $r_g$ values previously described.
Repulsion Factor (\textit{REP}) Computation:

// Scratch is a memoization array maintained in each chromosome's auxiliary data. This routine initializes it.
Scratch[256] = 0, 0, ... 0
Repulsion[256] = 0, 0, ... 0

FOR all points p1 in the array DO:
    FOR all points p2 within 9 cells of p1 DO:
        IF Threshold(p1) < 128 AND Threshold(p2) < 128 THEN
            dist = Distance(p1, p2)
            Scratch[min(Threshold(p1), Threshold(p2))] += exp(-.15*dist^2)
        ELSE IF Threshold(p1) > 127 AND Threshold(p2) > 127 THEN
            dist = Distance(p1, p2)
            Scratch[max(Threshold(p1), Threshold(p2))] += exp(-.15*dist^2)
        ENDIF
    ENDFOR
ENDFOR

sum = 0.0
FOR i = 0..127 DO:
    sum += Scratch[i]
    Repulsion[i] = sum
ENDFOR

sum = 0.0
FOR i = 255..128 DO:
    sum += Scratch[i]
    Repulsion[i] = sum
ENDFOR

sum = 0.0
denom = 0.0
FOR i = 2..253 DO:
    IF Repulsion[i] < 10 * Normalizer[i] THEN
        sum += Scale[i] * (Repulsion[i] / Normalizer[i])^2
    ELSE
        sum += Scale[i] * (7.7 + log(Repulsion[i] / Normalizer[i]))^2
    ENDIF
denom += Scale[i]
ENDFOR

RETURN exp(-1.0 * (sqrt(sum / denom) - 1))
3.3.2 Smoothness

The smoothness component of the fitness function measures the degree to which the halftone patterns produced by a threshold array are visually uniform, by calculating statistics based on the deviation of low-pass filtered halftone patterns from uniform gray images. The action of the low-pass filter simulates the action of the human visual system.

The smoothness component is computed as

$$SMOOTH(c) = e^{-0.65 \times \text{errsum}(c) - \text{maxerr}}$$

(3.3)

where

$$\text{err}_{x,y}(c, t) = \text{gray}(c, x, y, t) - \frac{t}{256}$$

(3.4)

$$\text{errsum} = \sum_{i=0}^{35} \sqrt{\frac{\sum_{x,y} \text{err}_{x,y}(c, 40 + i \times 5)^2}{\text{width} \times \text{height}}}$$

(3.5)

$$\text{maxerr} = \max_{x,y,i} \text{err}_{x,y}(c, 40 + i \times 5) \text{ for } i \in \{0 \ldots 35\}$$

(3.6)

and \( \text{gray}(c, x, y, t) \) is the gray value produced at the location \((x, y)\) in the halftone pattern generated by chromosome \(c\) at threshold \(t\). The gray value is determined by applying the following low-pass filter to the halftone pattern:

$$
\begin{bmatrix}
0.050 & 0.153 & 0.223 & 0.153 & 0.050 \\
0.153 & 0.472 & 0.687 & 0.472 & 0.153 \\
0.223 & 0.687 & 1.000 & 0.687 & 0.223 \\
0.153 & 0.472 & 0.687 & 0.472 & 0.153 \\
0.050 & 0.153 & 0.223 & 0.153 & 0.050 
\end{bmatrix}
\times \frac{1}{7.952}
$$

The above filter is derived from the Gaussian \( e^{-\frac{1}{8} x r^2} \).

The smoothness factor penalizes deviations from a uniform gray field and includes a particular penalty for the maximum error. The use of \( \text{maxerr} \) accounts for the fact that people tend to concentrate their attention and base their evaluations of quality on the region containing the worst errors in a halftoned image[5].
Smoothness Factor (*SMOOTH*) Computation:

```plaintext
sum = 0
maxerr = 0

FOR thresh = 40..215 BY 5 DO:
  levelsum = 0
  FOR all points p1 in the array DO:
    error = FilteredGray(p1, thresh) - thresh/256
    levelsum += error^2
  IF error > maxerr THEN
    maxerr = |error|
  ENDIF
ENDFOR
ENDFOR
sum += sqrt(levelsum / (width * height))
ENDFOR

RETURN exp(-0.65*sum - maxerr)
```

### 3.3.3 Partitioning

The partitioning component of the fitness function analyzes the halftone patterns produced by the threshold array and penalizes undesirable properties in the patterns at the 25%, 50% and 75% gray levels. Undesirable patterns at 50% gray include horizontal, vertical, and diagonal lines, checkerboards and variations on checkerboards. Undesirable patterns at 25% and 75% gray include adjacent minority pixels. The effect of this component is important in establishing quality halftone patterns at important gray levels.

The checkerboard pattern is an optimal solution for representing 50% gray on an ideal output device, however, arrays which generate this pattern produce unsatisfactory patterns at other gray levels around 50%, because any variation from a perfect checkerboard produces a visible defect in the pattern. For this reason, the partitioning component of the fitness function is used to ensure that the partitioning of pixels at 50% gray into those which are black and those which are white does not form a perfect checkerboard or a checkerboard variant with similar visual characteristics.

Specifically, the partitioning component examines the 5 × 5 cell neighborhood of
Figure 3-4: Illustration of pixel categorization at the 50% partitioning level

each cell at 50% gray and assigns the cell to one of three categories (see Figure 3-4 for illustrations):

**OK** Any neighborhood which is not in one of the Bad or Worst categories.

**Bad** Any neighborhood with same-color neighbors on two adjacent corners of the central pixel and an alternating pattern along the horizontal or vertical axis through the central pixel at 50% gray. Also, any neighborhood which features a line through the central pixel. This category penalizes variations on the standard checkerboard, such as a checkerboard with a 2:1 aspect ratio.

**Worst** Any neighborhood which has more than one horizontal, vertical or diagonal line, or has more than six or fewer than two pixels of the same color neighboring the central pixel at 50% gray. This category penalizes checkerboard patterns and patterns which are very far from yielding 50% gray even with a threshold of 128.

For the purposes of the categorization, a line is defined to be three or more horizontally or vertically contiguous pixels or four or more diagonally contiguous pixels.

Similarly, each cell is categorized as OK, Bad or Worst at 25% and 75% gray. A cell is considered Worst if any of its horizontal or vertical neighbors is the same color as the central pixel or if it has two or more diagonal neighbors of the same color as the central pixel. A cell is considered Bad if one of its diagonal neighbors is the same color as the central pixel. A cell is considered OK if it has no neighbors of the same color.
The value of $\mathcal{P}\mathcal{A}\mathcal{R}\mathcal{T}$ is computed as

$$\mathcal{P}\mathcal{A}\mathcal{R}\mathcal{T}(c) = e^{-3 \times \text{middle\_badness} - 5 \times \text{end\_badness}}$$

(3.7)

where

$$\text{middle\_badness} = \frac{\text{worst} + 0.2 \times \text{bad}}{\text{width} \times \text{height}}$$

(3.8)

$$\text{end\_badness} = \frac{2.0 \times (\text{badgrays} + 0.1 \times \text{okgrays})}{\text{width} \times \text{height}}$$

(3.9)
Partitioning Factor (\textit{PART}) Computation:

// bad and worst at 50\% gray
bad = 0
worst = 0

// bad and worst at 25\% and 75\% grays
badgrays = 0
worstgrays = 0

FOR all points \( p_1 \) in the array DO:

- \( \text{diag\_same} = \) number of immediate diagonal neighbors which are the same color as \( p_1 \) at 50\% gray
- \( \text{edge\_same} = \) number of edge-adjacent neighbors which are the same color as \( p_1 \) at 50\% gray
- \( \text{hline} = \) length of the horizontal line which includes \( p_1 \), within the 5x5 area centered on \( p_1 \)
- \( \text{vline} = \) length of the vertical line which includes \( p_1 \), within the 5x5 area centered on \( p_1 \)
- \( \text{d1line} = \) length of the diagonal line with negative slope which includes \( p_1 \), within the 5x5 area centered on \( p_1 \)
- \( \text{d2line} = \) length of the diagonal line with positive slope which includes \( p_1 \), within the 5x5 area centered on \( p_1 \)
- \( \text{halt} = \) number of pixels along the horizontal line through \( p_1 \) within the 5x5 area centered on \( p_1 \) which satisfy an alternating pattern
- \( \text{valt} = \) number of pixels along the vertical line through \( p_1 \) within the 5x5 area centered on \( p_1 \) which satisfy an alternating pattern

numlines = 0
IF hline >= 3 THEN numlines++
IF vline >= 3 THEN numlines++
IF d1line > 3 THEN numlines++
IF d2line > 3 THEN numlines++

Continued...
...PART continued from previous page

IF numlines > 1 OR hline > 3 OR vline > 3 THEN worst++
ELSE IF diag_same + edge_same > 6 OR
    diag_same + edge_same < 2 THEN worst++
ELSE IF halt > 3 AND d1line = 2 AND d2line = 2 THEN bad++
ELSE IF valt > 3 AND d1line = 2 AND d2line = 2 THEN bad++
ELSE IF numlines = 1 THEN bad++
ENDIF

IF p1 is black at 75% gray THEN
    IF any of p1's horizontal or vertical neighbors
        is black at 75% gray THEN worstgrays++
    ELSE IF more than one of p1's diagonal neighbors is black
        at 75% gray THEN worstgrays++
    ELSE IF only one of p1's diagonal neighbors is black
        at 75% gray THEN badgrays++
    ENDIF
ENDIF

IF p1 is white at 25% gray THEN
    IF any of p1's horizontal or vertical neighbors
        is white at 25% gray THEN worstgrays++
    ELSE IF more than one of p1's diagonal neighbors is white
        at 25% gray THEN worstgrays++
    ELSE IF only one of p1's diagonal neighbors is white
        at 25% gray THEN badgrays++
    ENDIF
ENDIF
ENDFOR

badval = (1.0 * worst + 0.2 * bad) / (width * height)
badval2 = 2.0 * (worstgrays + 0.1 * badgrays) / (width * height)

RETURN exp(-3 * badval) * exp(-5 * badval2)
3.4 Mutation Strategy

An important invariant which is maintained by all of the mutation operators is that for every threshold value, the number of cells of the matrix containing that value is preserved. This invariant guarantees that tone scale reproduction will be preserved on an ideal output device. Since all of the matrices are created with linear tone scale and all of the mutation operators preserve this property, all arrays generated will have linear tone scale.

3.4.1 Order-Based Crossover

A standard crossover operator cannot be used with the chosen chromosome representation because it could violate the linearity invariant which ensures proper tone scale reproduction. An order-based crossover operator which combines values from a single parent with ordering information from both parents is used instead[2].

The Order-Based-Crossover routine combines two threshold arrays by ordering the values within a rectangular region of one parent array in the order of the values in the corresponding region of the second parent array. All values in the offspring array outside of that region are identical to the primary parent, and the values inside of that region consist of the values from the primary parent reordered so that the order in which the cells change from black to white is identical to the order in the region of the second parent.

3.4.2 Point Swaps

Single-point swaps provide small chromosome mutations. Each swapping mutator chooses two cells of the threshold array satisfying some criteria and swaps the values of the chosen cells. There are five point-swap mutators:

Point-Swap-Same: Swaps the values of a randomly chosen pair of cells whose thresholds are both below 0.5 or both above 0.5.
Point-Swap-Close: Swaps the values of a randomly chosen pair of cells whose thresholds differ by no more than 8%.

Point-Swap-Different: Swaps the values of a randomly chosen pair of cells where one threshold is below 0.5 and the other is above 0.5.

Smoothness-Swap: Swaps the value of a cell which neighbors a cell which performs poorly on the smoothness component of the fitness function with the value of another cell which will improve the smoothness score for each. The pairs are found by repeated random trials with a goal which relaxes at each iteration.

Redistribution-Swap: Swaps the values of a pair of cells in such a way that the tone scale of a 4 × 4 region containing each cell is improved (made more linear). The linearity is determined by a course 16-bin histogram of the area. Since the area contains 16 cells, each bin in a histogram of a perfectly linear region should contain a single cell. If a region is not perfectly linear, than at least one bin will contain more than one cell and at least one bin will contain no cells. The mutator swaps a cell from the most crowded bin with a cell which will fill its empty bin. The other cell is taken from a region in which the cell with the needed value is in a crowded bin.

3.4.3 Point Movement

Movement is a type of mutation which exhibits spatial locality: points are swapped only with nearby points. Movement mutations provide a different mode of fine adjustment to the threshold array than the Point-Swap-Close mutator. There is one undirected and one directed movement mutator:

Point-Movement: Randomly swaps a point with another point fewer than three cells away in the threshold array.

Repulsion-Movement: Swaps a point with another point up to four cells away. The direction of motion is determined by a repulsion calculation which takes
into account the configuration of thresholds in the neighborhood around the selected point.

### 3.4.4 Operator Reweighting

Operator reweighting is a technique for dynamically altering the frequency with which mutators are applied in accordance with the efficiency the mutator has demonstrated during recent applications. Over the course of an entire run, operator reweighting allows the algorithm to favor the mutators which are most successful with the population at the moment. It allows mutators to be promoted while they are useful and inhibited when they become unproductive.

Dynamic operator reweighting also provides the algorithm designer with a tool to analyze and quantitatively compare the performance of different mutation strategies.

When using dynamic operator reweighting, the performance of mutation operators is monitored and the more efficient mutators are applied more frequently. Mutators are selected according to a roulette wheel in a fashion similar to parent selection. Each population maintains a set of weights which reflect the fitness of the offspring created by each mutator. These weights are used to determine the selection frequencies for the operator roulette wheel.

Operator reweighting is accomplished in this implementation by associating each operator with a weight and an accumulator. Each time a new chromosome is inserted into the population, the accumulator is incremented by an amount which reflects the quality of the new chromosome relative the rest of the population. Specifically, the accumulator is not incremented if the chromosome is not in the top 10% of the population. If the chromosome does rank in the top 10% its accumulator is incremented by \( \frac{0.1}{1.0 + 50.0 \times \frac{\text{rank}}{\text{population size}}} \). If the chromosome is the highest ranking individual, the accumulator is incremented by an additional amount \( \max(0.1, 1000 \times (\text{fitness}_0 - \text{fitness}_1)) \), where \( \text{fitness}_0 \) is the fitness of the chromosome (of rank 1) and \( \text{fitness}_1 \) is the fitness of the chromosome of rank 2.

Every ten generations, the operator weights are recalculated from the values of the accumulators and the accumulators are reset to 0. The operator weights are
calculated by first decreasing the weights by a factor of 0.75, but they are never allowed to decrease below 5. An invariant is maintained such that the sum of the operator weights will always equal 1000. Each operator’s weight is incremented by an amount equal to the operator’s accumulator value normalized to maintain the invariant on the sum of the weights.

3.5 Important Optimizations

Several important optimizations drastically improve the efficiency of this intensive computation, especially for very large threshold arrays.

3.5.1 Memoization of Fitness Calculations

Caching partial results of the fitness evaluation can significantly reduce the number of redundant calculations performed. For example, swapping two values which occupy the same partition (i.e., x/64 = y/64) will not alter the value of the \texttt{PART} function, so recalculating that value after such a mutation is wasteful. Similar relationships can be derived for the other mutators, but the strategy is most successful with the \texttt{PART} function.

The \texttt{REP} function does not yield to this simple caching scheme. Almost any swap will require a recalculation of the repulsion component of the fitness function. Although this recalculation cannot be avoided, it is possible to perform a cheaper calculation to account for the effect of the swap without repeating the entire repulsion component calculation by updating the partial results memoized in the chromosome’s auxiliary data and continuing the calculation with the updated values.

3.5.2 Hints for Mutators

The effort of calculating the fitness evaluation can be used to provide hints to the mutation operators about troublesome cells in the threshold arrays. For example, the calculations of the \texttt{REP} component of the fitness function can provide valuable
Updating Repulsion Factor ($\mathcal{R}_E$) After A Point-Swap:

Scratch[256] = // values from previous REP factor calculation
              // memoized in the chromosome's auxiliary data
Repulsion[256] = 0, 0, ... 0

Swap(loc1, loc2) // undo swap temporarily

// remove contribution of val1 at original location
p1 = loc1
FOR all points p2 within 9 cells of p1 DO:
   IF Threshold(p1) < 128 AND Threshold(p2) < 128 THEN
      dist = Distance(p1, p2)
      Scratch[min(Threshold(p1), Threshold(p2))] +=
         -2.0 * exp(-0.15 * dist^2)
   ELSE IF Threshold(p1) > 127 AND Threshold(p2) > 127 THEN
      dist = Distance(p1, p2)
      Scratch[max(Threshold(p1), Threshold(p2))] +=
         -2.0 * exp(-0.15 * dist^2)
   ENDIF
ENDFOR

// remove contribution of val2 at original location
p1 = loc2
FOR all points p2 within 9 cells of p1 DO:
   IF Threshold(p1) < 128 AND Threshold(p2) < 128 THEN
      dist = Distance(p1, p2)
      Scratch[min(Threshold(p1), Threshold(p2))] +=
         -2.0 * exp(-0.15 * dist^2)
   ELSE IF Threshold(p1) > 127 AND Threshold(p2) > 127 THEN
      dist = Distance(p1, p2)
      Scratch[max(Threshold(p1), Threshold(p2))] +=
         -2.0 * exp(-0.15 * dist^2)
   ENDIF
ENDFOR

Continued...


...REP update continued from previous page

Swap(loc1, loc2) // redo swap

// replace contribution of val1 at new location
p1 = loc2
FOR all points p2 within 9 cells of p1 DO:
  IF Threshold(p1) < 128 AND Threshold(p2) < 128 THEN
    dist = Distance(p1, p2)
    Scratch[min(Threshold(p1), Threshold(p2))] +=
      2.0 * exp(-0.15 * dist^-2)
  ELSE IF Threshold(p1) > 127 AND Threshold(p2) > 127 THEN
    dist = Distance(p1, p2)
    Scratch[max(Threshold(p1), Threshold(p2))] +=
      2.0 * exp(-0.15 * dist^-2)
  ENDIF
ENDFOR

// replace contribution of val2 at new location
p1 = loc1
FOR all points p2 within 9 cells of p1 DO:
  IF Threshold(p1) < 128 AND Threshold(p2) < 128 THEN
    dist = Distance(p1, p2)
    Scratch[min(Threshold(p1), Threshold(p2))] +=
      2.0 * exp(-0.15 * dist^-2)
  ELSE IF Threshold(p1) > 127 AND Threshold(p2) > 127 THEN
    dist = Distance(p1, p2)
    Scratch[max(Threshold(p1), Threshold(p2))] +=
      2.0 * exp(-0.15 * dist^-2)
  ENDIF
ENDFOR

Continued...
...REP update continued from previous page

    sum = 0.0
    FOR i = 0..127 DO:
        sum += Scratch[i]
        Repulsion[i] = sum
    ENDFOR

    sum = 0.0
    FOR i = 255..128 DO:
        sum += Scratch[i]
        Repulsion[i] = sum
    ENDFOR

    sum = 0.0
    denom = 0.0
    FOR i = 2..253 DO:
        IF Repulsion[i] < 10 * Normalizer[i] THEN
            sum += Scale[i] * (Repulsion[i] / Normalizer[i])^2
        ELSE
            sum += Scale[i] * (7.7 + log(Repulsion[i] / Normalizer[i]))^2
        ENDIF
        denom += Scale[i]
    ENDFOR

    RETURN exp(-1.0 * (sqrt(sum / denom) - 1))
Figure 3-5: Use of staged computations to generate large threshold arrays.

information about which cells make the least contribution to the fitness function so that the mutators can concentrate on improving those cells. Similarly, the calculations of the SMOOTH component can provide hints about the cells which give rise to the largest deviations from the intended gray level during halftoning. These hints are passed to the mutation operators through the auxiliary data associated with each chromosome.

3.5.3 Seeding and Staged Computation

Genetic optimization may require many generations to produce quality threshold arrays from the pool of random threshold arrays with which the computation begins. The bulk of those generations may be avoided by seeding the initial population with arrays which are already of moderately high quality.

Figure 3-5 shows \( \mathcal{N} \) computations can be staged to generate good seeds for later computations. The initial stage can generate several small arrays, which can be combined to produce seeds for a later stage. The small arrays are combined in a pattern which tries minimize the number of repeated edge matchings, as these will lead to regular artifacts in the seed array. The seeds produced in this way are of high quality except along the seams between the smaller arrays in the mosaic. Smaller arrays may also be computed using other, faster methods such as the void-and-cluster method. This staging technique can reduce the time required to generate large arrays by up to 90%.
Chapter 4

Results

This chapter will examine the threshold arrays produced using this method and compare their visual and spectral properties to void-and-cluster blue noise masks and to error diffusion. The role played by each fitness function component in determining the characteristics of the generated array, and the efficiency of each mutation operator will be explored.

4.1 Threshold Arrays

Figure 4-1 compares grayscale ramps halftoned using a threshold array generated using the genetic method described in this thesis with the reference image, the image halftoned using void-and-cluster thresholds, and the image halftoned using the Floyd and Steinberg error diffusion technique[4]. The GA thresholds provide a blue noise pattern comparable to that of the void-and-cluster thresholds in the highlights and shadows, but are less grainy in the regions around 25% and 75% gray and have fewer low-frequency artifacts around 50% gray. Both the void-and-cluster and the GA thresholds are free of the contours and textures which mar the image halftoned with the error diffusion technique. Those textures can be eliminated or reduced by introducing random variations into the error diffusion process.

Figure 4-2 compares the test image ‘Lena’ halftoned using the same techniques. Again, the error diffusion technique introduces unsightly artifacts which are not
Figure 4-1: Comparison of grayscale ramps using GA thresholds to images using void-and-cluster thresholds and error diffusion.
Figure 4-2: Comparison of test photo using GA thresholds to images using void-and-cluster thresholds and error diffusion.
present in either the void-and-cluster image or the GA threshold image. The error diffusion image is sharper than either of the others, and the GA thresholds image is slightly sharper than the void-and-cluster image.

4.2 Spectral Properties

Figure 4-3 shows spectra for 50%, 60% 75% and 90% gray patterns halftoned using blue noise masks created by the void-and-cluster method and the genetic method and patterns halftoned by error diffusion. The spectral estimates were generated by first calculating the two-dimensional Discrete Fourier Transform of a $128 \times 128$ halftoned pattern and then calculating the annular average of the squared and normalized frequency samples in order to obtain a one-dimensional power spectral density estimate. The DC term has been omitted to show the remaining data at a readable scale.

Because of the periodic nature of threshold arrays, it is difficult to obtain power spectral density estimates with low variance. The computation of Figure 4-3 used thirty-four annuli to give both informative resolution in the frequency domain and acceptable variance in the frequency samples.

4.3 Evolution of the Population Over Time

Figure 4-4 shows the evolution of each of the three components of the fitness function over 5000 generations of a typical run. Each component shows gradual improvement over time, with the bulk of the improvement occurring during the first 2500 generations.

Figure 4-5 shows a grayscale ramp halftoned with the most fit threshold arrays in the population as the optimization progresses. There is relatively little visible improvement in the image during the final half of the run; most of the improvement during that period is localized to the regions between about 70% and 93% gray and between about 7% and 30% gray.
Figure 4-3: Comparison of spectral properties of blue noise masks generated using the genetic technique with masks created using the void-and-cluster method and patterns produced by error diffusion. The error diffusion spectrum is not shown for 50% gray because it consists of an impulse at $f_g = \frac{1}{\sqrt{2}}$ which dwarfs the spectral components of the other two images.
Figure 4-4: Fitness component curves during a typical run of the genetic optimizer.

4.4 Running Time

Figure 4-6 compares the time required to generate threshold arrays of different sizes using the genetic technique presented in this thesis and the void-and-cluster method. The timings were performed on a 486DX4/100 personal computer. The timings show that the constant factors involved in the genetic computation dwarf the time requirements of the void-and-cluster method for small arrays. For arrays larger $128 \times 128$, staged genetic optimizations can become significantly faster. It should be noted, however, that the genetic algorithm implementation used in these timings was highly optimized, whereas the void-and-cluster implementation, though efficient, was not.

The staging technique which generates and combines several small arrays to generate high-quality seeds for later runs is a very successful strategy which takes advantage of the properties of genetic search techniques. By generating four $16 \times 16$ arrays and combining them into a $128 \times 128$ seed array, the threshold array demonstrated in Figure 4-7 was generated in about 7 days using this technique. The images show residual artifacts along the seams, indicating that additional optimization or better mutation strategies could further improve the quality of the threshold array. An unstaged calculation would require about 30 days to calculate a blue noise mask of this size and the void-and-cluster method would require 60 days.
Figure 4-5: Comparison of grayscale ramps using GA thresholds produced over the course of a typical run.
Figure 4-6: Elapsed time to generate threshold arrays of size 16, 32, 64 and 128 using a genetic algorithm and the void-and-cluster method.

4.5 Effects of Fitness Function Components

Figure 4-9 shows the output of threshold arrays generated by optimizing only the repulsion component ($REP$) of the fitness function. The repulsion component produces good blue noise patterns in the highlights and shadows, but suffers from low-frequency artifacts around 25% and 75% gray.

Figure 4-10 shows the output of threshold arrays generated by optimizing only the smoothness component ($SMOOTH$) of the fitness function. The smoothness component produces slightly fewer artifacts in the the 25% and 75% grays and throughout the midtones in general, however it does nothing to control the patterns in the highlights and shadows.

4.6 Mutation Operator Fitness

Figure 4-11 shows the fitness of each mutation operator over the course of a typical run.

The Order-Based-Crossover operator performs consistently well during the entire run.

Of the four undirected operators, Point-Swap-Different performs the worst. Its
Figure 4-7: $128 \times 128$ threshold array generated using staged computation.
Figure 4.8: Threshold array generated by optimizing all of the components of the fitness function.
Figure 4-9: Threshold array generated by optimizing only $\mathcal{R}_F(c)$. 

a) 90% gray
b) 75% gray
c) 50% gray
d) Grayscale ramp
Figure 4-10: Threshold array generated by optimizing only $SMOOTH(c)$.
Figure 4-11: Mutation operator fitness curves during a typical run of the genetic optimizer.
fitness falls off early in the run and never recovers. The Point-Swap-Same operator performs well during the initial portion of the run but soon falls into disuse after the coarse mutations it performs become more disruptive than beneficial. As the Point-Swap-Same operator wanes, the Point-Swap-Close operator becomes active, making fine adjustments to the threshold levels at each cell. The Point-Movement operator shows great variability: long periods of dormancy punctuated by sporadic periods of very high fitness. This pattern suggests that the mutations provided by the Point-Movement operator are not often beneficial, but when they are they provide significant improvements in the overall fitness of the threshold array.

The three directed mutators show high fitness during the initial portion of the run but eventually lapse into mediocre performance for the duration of the run. The Repulsion-Movement operator performs well initially but falls into a pattern of poor performance interspersed with short bursts of beneficial mutations. The Smoothness-Swap mutator quickly falls into a pattern of steady mediocre performance, neither good nor bad. The Redistribution-Swap operator follows a similar trend, though it usually performs slightly worse than the Smoothness-Swap.
Chapter 5

Conclusions

The primary thrust of this work has been to develop a flexible mechanism for creating threshold arrays for digital halftoning. The threshold arrays are "bred" using genetic search techniques which explore the solution space based on a fitness function which describes the desired properties of the threshold array. The threshold arrays produced in this thesis used a fitness function which rewarded even distribution of points at each gray level and penalized deviation from the desired gray value and low-frequency artifacts in the halftoned image.

The genetic method produces blue noise masks of quality comparable to that of arrays produced using the void-and-cluster method of Robert Ulichney[10]. The genetic method is much slower than the void-and-cluster method for small arrays but is somewhat faster for large arrays. The ability to breed solutions from an initial seed array allows genetic computations to be staged in a way that scales up to very large threshold arrays without incurring the high cost of optimization from a large random seed.

The most significant aspect of this work is the flexibility of the approach. It is a straightforward task to replace the fitness function used in this thesis with an improved fitness function that incorporates a printer model or an improved perceptual model. This flexibility allows the genetic technique to produce arrays which are better tuned to specific applications and marking technologies.

Finally, there is great potential for improvement of this method. Better fitness
functions, perhaps using threshold array representations based on nearest-neighbor graphs, more intelligent mutation strategies, and mutators which concentrate on the seams produced during staged computations could all yield improved efficiency and array quality.
Bibliography


