Hierarchical Graph Color Dither

Alejo Hausner

Department of Computer Science,

University of New Hampshire,

Durham, NH, USA

Web: www.cs.unh.edu/~ah

Email: ah@cs.unh.edu

Keywords: Halftoning, Ordered Dither, Hierarchical Graph Coloring
ABSTRACT

This paper studies dispersed-dot threshold matrices, and examines an aspect of these matrices that has not been considered before, the distance between consecutive thresholds: suppose the threshold matrix is treated as a collection of numbers, where each number has a position in space, and suppose the numbers are visited in increasing order; what is the distance in space between pairs of consecutive numbers visited?

We observe that in Bayer’s matrices this consecutive distance is always large. We hypothesize that this large consecutive distance is an important quality for good dispersed-dot threshold matrices. To study the hypothesis, we generate matrices that have this quality. We derive a method that can generate such matrices, by solving a more general problem: given an arbitrary set of points on the plane, sort them into a list in which consecutive points are far apart in space. Our solution relies on coloring the nearest-neighbor graph, hierarchically.

The method does reproduce Bayer’s dispersed-dot dither matrices under some settings, and furthermore can produce matrices of arbitrary dimensions. Multiple similar matrices can be created to minimize repetitive artifacts that plague Bayer dither, while retaining its parallelizability. The method can also be used for halftoning with points on a hexagonal grid, or even randomly-placed points. It can also be applied to artistic dithering, which creates a dither matrix from a motif image. Unlike in the artistic dither method of Ostromoukhov et al, the motif image can be arbitrary, and need not be specially constructed.
1. INTRODUCTION

The human eye can perceive a large range of colors, but most printing devices are limited to only a few colors. To create the illusion of many colors, printing technology uses halftoning. Halftoning methods arrange dots of a few colors into patterns that the viewer’s eye can blend into the desired color.

In the simplest case, the device uses black ink on white paper. To produce a printable version of an image, each pixel’s color is set to either black or white. This process of restricting a large range of colors to a small range (in this case, two colors) is called quantization. To quantize a pixel, its color compared is to a threshold value. If the pixel’s brightness is greater than the threshold, it is set to white; if less, it is set black.

The two most common halftoning methods are error diffusion and ordered dither. Both methods use quantization, but in different ways. In error diffusion, the threshold value is always 0.5, halfway between black (0.0) and white (1.0). For example, if the pixel was 0.7, it will be quantized to 1.0. After quantization, the change in pixel value, also called the error (in this example, the error is 0.7-1.0=-0.3), is added in varying proportions to the pixel’s yet-to-be-processed neighbors (the error is “diffused” to the neighbors). In error diffusion, pixels are usually processed row by row, and the unvisited neighbors receiving the error are below and to the right of the current pixel. The pixels must be processed serially; pixels in lower rows may only be quantized after higher rows have been processed.

Ordered dither also works by quantization, but does not keep the threshold value fixed. Instead, each pixel may potentially have a different threshold. For a large image, computing every threshold for every pixel is usually very time-consuming. To save time, usually these thresholds are computed once offline, for a relatively small square block of pixels. This block is the threshold matrix or dither
matrix. The threshold matrix is repeated across the image, just like square tiles being laid on a floor. Ordered dither can process the pixels in any order, because no error is transferred between pixels, unlike in error diffusion. Hence dithering is easily parallelized and can be made extremely fast.

Because ordered dither repeats the same block of thresholds across the image, a human observer will often perceive an obvious repetitive pattern. Error diffusion does not suffer from this particular visual artifact. Despite this drawback of ordered dither, its speed makes it attractive. Even more interesting, the very fact that the dither matrix is repeated can be an advantage, because special matrices can be used to create artistic effects, producing images made up of many tiny artist-chosen motifs.

Figure 1a shows a small 4×4 dither matrix. This is a dispersed-dot dither matrix. Suppose the thresholds in the matrix are listed in increasing order: \((0/16, 1/16, 2/16, \cdots, 15/16)\). Where in the matrix will these numbers occur? Notice that consecutive thresholds in this list are never near each other in the matrix: 0/16 is not adjacent to 1/16, which is not adjacent to 2/16, and so on.

Now consider what happens when this threshold matrix is used to halftone a 12×12-pixel image. To cover this image, the matrix must be repeated three times horizontally and vertically, as shown in fig. 1b. The input original image (not shown) is a solid block of gray, where every pixel has value 4/16. Fig. 1c shows the resulting output halftone. Notice the pattern of white pixels where thresholds 0, 1/16, 2/16 and 3/16 occur (at those pixels, the pixel color is greater than or equal to the threshold, and therefore is set white). In other words, the output halftone consists of dispersed dots; there are no solid clusters of white pixels. That is why this is a dispersed-dot matrix.
1.1. Hypothesis

We hypothesize that the two properties just described are intimately connected. In other words, if consecutive thresholds are far apart, then output halftones will have dispersed dots. To study this conjecture, we will build matrices that have large separations between consecutive thresholds, and study the types of halftones they yield.

1.2. Graph Ordering

This paper presents a new generalization of ordered dither. The generalization arises by focusing on the order of the entries in the dither matrix. A dither matrix is defined by the visitation order of its entries. The matrix can also be seen as defining a graph, with pixels as vertices and edges joining neighbors, and thus visitation is equivalent to creating an ordering for the graph’s vertices. Thus dithering is an instance of a more general problem: graph ordering. A vertex ordering of a graph is a listing of its vertices in some order.

To create this ordering, a new method is presented, based on graph coloring. Vertices in a graph are colored, and then each subset of vertices with the same color is colored in turn, recursively. Each vertex has a label, and coloring adds a digit to the label. When the process is complete, each
vertex’s label is a multi-digit string which gives the vertex’s place in the ordering. The coloring algorithm is the main contribution of this paper.

By studying dithering as a graph-ordering problem, many interesting results are obtained. First, dithering can now be applied to any collection of points, not just points on a regular pixel grid. Second, Bayer’s ordered dither matrices are obtained as a special case, when the points lie on a $2^n \times 2^n$ square grid. Bayer-like dither matrices of arbitrary dimensions are also obtained. Further, by varying the coloring algorithm, many Bayer-like dither matrices of the same dimensions can be produced, which can be used to reduce the artifacts that arise from repeating the same matrix. Finally, it is simple to obtain dither “matrices” for other arrangements of points, such as hexagonal grids or random points.

The graph-ordering approach also leads to improved artistic halftoning. This kind of halftoning replicates a small motif across a larger image, producing attractive results. Previous approaches require specialized motifs, restricting the range of possible halftones. With graph-ordering, any small image can be used as motif.

1.3. Contributions

The main contribution of this paper is a hierarchical graph coloring algorithm, which generalizes dispersed-dot dither to arbitrary collections of points. This algorithm has many applications, including the following:

- It solves the general problem of ordering a collection of points such that consecutive points in the order are far apart.

- It provides a general-purpose technique for creating a wide range of ordered-dither matrices:
- Bayer’s ordered-dot threshold matrices, when the points in the collection lie on a square grid.

- threshold matrices of arbitrary dimensions, not just $2^k \times 2^k$.

- threshold matrices for points in a hexagonal grid.

• It can be used for halftoning with randomly-placed points.

• It can produce multiple variants of a threshold matrix, useful for avoiding repetitive artifacts encountered by repeating a single matrix.

• By varying the graph-coloring algorithm used, it can produce threshold matrices with a range of properties, from highly structured ones that resemble Bayer’s dither, to ones that resemble random thresholding.

• It easily can use any gray-scale image as a repetitive motif useful for artistic halftoning.

Of course, some of these results (arbitrary-size matrices, hexagonal-grid dithering, randomly-placed points) have been obtained in other ways. However, the present method is general enough that many of these previous results can be obtained with a single algorithm, either by varying either the algorithm’s inputs or some of its internal settings.

The paper also studies the relative merits of several graph-ordering algorithms, using both halftone gray ramps and point-spread distributions. A final contribution is an explanation for why certain geometrical arrangements of dots cannot be halftoned in a cluster-free manner for some gray levels. In particular, pixels arranged on a hexagonal grid will always exhibit clusters for a 50% gray.
1.4. Outline

The rest of this paper is organized as follows: Section 2 discusses related previous work in halftoning and graph ordering. The hierarchical graph coloring (HGC) approach is described in section 3, and several variations of the HGC algorithm are discussed in section 4. The distribution of consecutive-point separations is discussed in section 5, and test images using the resulting dither matrices are considered in section 6. HGC is applied to artistic halftoning, with results in section 6.5. The paper closes with a summary and conclusion in section 7, including suggested future work.

2. BACKGROUND AND RELATED WORK

Halftoning continues to be an active area of research. A thorough review of past halftoning techniques would be impractical, but several good works on the subject should be emphasized: Ulichney’s book\(^2\) presents an excellent review of “classical” halftoning, including dithering and error diffusion. Newer techniques are covered in two collections edited by Eschbach\(^3,4\). A more recent work by Kang\(^5\) also includes newer topics like color halftoning and combining ordered and random dithering. In this section, we will focus on past work related to our problem; a curious reader may refer to the above texts for more details.

Halftoning processes an input image, and produces an image of the same dimensions, which should visually resemble the original, but uses a palette with fewer colors. In the binary case (black and white halftone), a palette with only two colors is used. The simplest form of halftoning is fixed thresholding: each pixel is replaced by the color closest to it in the palette. This approach cannot reproduce gradations of gray in the input image.

Dispersed-dot ordered dither was introduced by Bayer\(^7,8\) as a way to improve on fixed thresholding. To reproduce tones of gray, black dots are spread using a \(2^k \times 2^k\) dither matrix of thresholds.
The matrix is repeated across the input image, and thresholding is applied again. A $4 \times 4$ Bayer matrix is shown in Fig. 1a. A Bayer matrix has the property that, if its entries are visited in order of increasing value, the positions of consecutively visited entries will be relatively far apart. Thus, in a region where the input image has constant grey value, all pixels up to that grey value will be set white, and these white pixels will be spread apart. Spreading points apart helps the eye ignore the dots themselves, and perceive the overall combination of the pixels, which is the input grey value.

In contrast, in clustered-dot ordered dither, consecutive values in the threshold matrix occupy adjacent matrix entries, as in Fig. 1d. In an input region of constant grey value, the pixels set to black will be clustered into larger round dots, which is suitable for printing devices where small dots cannot be arbitrarily placed without merging.

Bayer’s dither produces obvious repetitive artifacts, but it is possible to use randomization to create many slightly different dither matrices, and use a different matrix for each square tile. No single matrix is repeated, and repetitive effects are thus reduced.

Error diffusion also avoids repetitive tiling artifacts, but it is not amenable to parallelization. Ordered dither is simpler to implement—once the dither matrices have been computed—and is easily parallelized. In particular, Ulichney’s void-and-cluster approach produces good ordered dither matrices for various tone values. Like the method presented here, the void-and-cluster technique can work with pixels in arbitrary positions, and is not restricted to a regular grid. Another approach avoids a regular grid, by starting with a set of points in random positions, and varying the sizes of the points’ Voronoi regions to convey varying gray levels in the source image.

The dither methods presented above all assume that the images are gray-scale only, but ordered dither can also be used for color reproduction. Ostromoukhov et al. present a general approach
to adapt ordered dither to color reproduction. They also show how ordered dither matrices can be built to create artistic screens. In an artistic screen, the dither pattern is made evident, and is emphasized to show a repeated decorative pattern superimposed on the input image (fig. 26 shows an example). This differs from ordinary dithering, where efforts are made to make the dither pattern inconspicuous. One limitation of the method presented by Ostromoukhov is that the artistic screen image must be carefully chosen to include as many grey levels as possible. Binary (black and white) motif images do not yield good results without additional processing. The present work overcomes this limitation.

The present method uses graph ordering techniques, and some attention must be paid to previous work on them. Almost all work on graph ordering focuses on ordering nodes to ease the process of partitioning a triangle mesh, breaking it up into pieces for parallelization in fluid dynamics computations. An example of this application is by Schamberger et al., who present a hierarchical division approach to ordering the nodes. As far as the present author is aware, graph ordering has not been previously applied to the problem of halftoning.

We will use the term hierarchical graph coloring (HGC) to describe our approach. This term has been applied to algorithms for register allocation in compiler design. Despite the possible confusion, we use the same term, because it describes our process so aptly.

3. APPROACH

Our method extends dispersed-dot dither, so we will first consider some properties of dither matrices. A dither matrix is a rectangular $r \times c$ array of thresholds (usually, $r = c = 2^k$, a power of 2). Each threshold is a fraction $i/(rc)$, with $i$ ranging from 0 to $rc - 1$. Equivalently, each fraction can be multiplied by $rc$, yielding a matrix of natural numbers. The positions of these natural numbers
determine the properties of the dither matrix. When considering dither matrices, we will focus on one particular question: if the thresholds are listed in increasing order, in what order will the cells of the matrix be visited? More specifically, what is the spatial separation (the ordinary Euclidean distance) between consecutive cells visited?

Two main classes of dither matrices are common: clustered-dot and dispersed dither (see Fig. 1a,d). In a clustered-dot dither matrix (Fig. 1d), consecutive numbers occupy adjacent or nearby cells, and a sequential enumeration of the cells will visit them in a spiral fashion, starting from the center. In contrast, in a dispersed-dot matrix, consecutive numbers \((i, i+1)\) occupy cells that are far from each other. Moreover, in a good dispersed-dot matrix, all consecutive triples \((i, i+1, i+2)\) and even longer sequences \((i, i+1, i+2, i+3)\) will all be far apart from each other.

We will focus on dispersed-dot dither. Our extension arises by considering a more general problem: given a collection of \(n\) points on the plane, in what order should they be visited such that consecutive dots are far away from each other? Again, we also want large separation not just between any two points in the sequence, but also in all runs of 3, 4, or more points.

Although we began this section by considering a rectangular grid of points, this more general problem exists for any given any set of points, whatever their spatial positions. For any set of points, some ordering exists such that consecutive points are far apart.

This more general problem is more difficult, but turns out to be worth addressing, because its solution also yields benefits. It provides a general way to reproduce several existing methods, and moreover, provides improvements on existing artistic halftoning approaches.
3.1. Graphs

The problem can be solved by coloring graphs. Graphs are used extensively in our approach, so before describing our solution, we will first consider some properties of graphs. This section describes what graphs are, and focuses on those properties that are used by our algorithms.

**Graphs**: In its most general definition, a graph consists of two sets: a set of *vertices*, and a set of *edges*. Each vertex is an object of some kind, and each edge “connects” two objects. For example, in Figure 2, consider the graph \((V, E)\) where \(V\) is this set of vertices: \(V=\{A, B, C, D\}\) and \(E\) is this set of edges: \(E=\{(A,B), (B,C), (A,C), (B,D), (C,D)\}\). Notice that each element of \(E\) (each edge) is a pair of vertices (the two vertices that the edge connects). This graph can be drawn in several ways, as seen in Figure 2a and 2b. If two vertices are joined by an edge, they are said to be *adjacent*. In this graph, A and B are adjacent (the pair \((A,B)\) is in the set \(E\)), but A and D are not.

**Euclidean Graphs**: The words “vertex” and “edge” are borrowed from geometry. They suggest that vertices are points, and edges are line segments. In other words, it suggests that vertices have position. This need not be so: for example, a graph may consist of a set of people (the vertices), and social relationships between them (the edges). Graph vertices can also be abstract entities.
without an actual position. However, if each vertex of a graph does have a known position \((x, y)\) in space, then the graph is called “Euclidean”. Each drawing in Figure 2 is thus a Euclidean graph.

**Planar Graphs** : Some graphs are such that their vertices can be given positions on the plane (the graph can be made Euclidean), and then their edges can be drawn in such a way that no two edges cross. To draw these edges, either straight-line segments or curves may be used, as long as no edges cross. If a graph can be drawn in this way, it is called *planar*. Most of the graphs considered in this paper are planar. For example, the graph in Figure 2 is planar, because, as fig. 2b shows, it can be drawn with no crossed edges.

**Neighbor Graphs** : Given a set of points on the plane, one can build a *triangulation* by connecting neighbor points with line segments, in such a way that the plane is divided into triangles. For a given set of points, many triangulations can be constructed. Figure 3 shows two triangulations for the same set of six points. The triangulation in Figure 3b is a *delaunay* triangulation. Every delaunay triangulation is a planar graph. Its vertices are the points, and its edges are the triangle edges. The defining property of a delaunay triangulation is this: for each triangle, the circle passing through its corners will contain no other vertex, as seen in fig. 3c. In contrast, the triangulation shown in fig. 3a does not obey this property: the circle passing through B, E, and F (the corners of a triangle) contains vertex A. The delaunay triangulation is used often in the algorithms presented below. We will also call it a “neighbor graph”, because the vertices joined by its edges are usually close together in space. Notice that the triangulation in fig. 3a is not a delaunay triangulation, and contains edges (B,F) and (B,D) which join relatively distant vertices, so it can’t be considered a neighbor graph.
Graph Coloring : The present methods work by coloring graphs. In a graph coloring, each vertex is given a color. In a proper vertex coloring, the colors cannot be arbitrary: if two vertices
Figure 5. a) a planar graph, with an improper vertex coloring (several adjacent nodes get the same color); b) a proper coloring using 5 colors; c) a coloring that uses 3 colors (the least possible for this graph).

are adjacent (are connected by an edge), they must have different colors. The word “color” may suggest to the reader visible colors like red or blue, but a color in our case is simply a label. In particular, we will consider a color to be simply a positive integer. Figure 5 shows the same graph colored in several ways. The first coloring is not proper (there are several cases where adjacent nodes have the same color). The second coloring is proper and uses five colors, while the third is also proper, but uses only three. The figure shows that the same graph can be colored in many ways. In fact, even fairly small graphs can be properly colored in a huge number of ways. The large number of proper colorings provides flexibility that our algorithms take advantage of.

k-Colorability : If a graph can be properly colored with k colors, it is said to be \textit{k-colorable}.

For example, the graph in Figure 5 is 3-colorable, because three different colors are used. For any graph, there is a minimum k. For example, for points on a square grid, the 4-orthogonal neighbor graph is 2-colorable, while the 8-neighbor graph is not: the 8-neighbor graph needs at least 4 colors for a proper coloring, as seen in Figure 6c.
Figure 6. Both a) the 4-orthogonal and b) 4-diagonal pixel graphs are 2-colorable, but c) the 8-neighbor graph needs 4 colors. d) For points in a checkerboard pattern, the 4-orthogonal neighbor graph is also 2-colorable.

A simple yet important result of graph theory is the four-color theorem (although its proof is somewhat controversial, because it involves too many cases to be verified manually, and hence requires trusting a computer algorithm to check them all). The theorem states that, for any planar graph whatsoever, four colors suffice. In other words, every planar graph is 4-colorable. This means, for example, that in any political map, regions (countries or states) may colored in such a way that any two regions sharing a border will receive different colors. This is true because a political map is structurally the same as a planar graph.

3.2. Key Idea

The main idea underlying the algorithms presented here is that coloring a graph increases vertex separation. For example, consider the neighbor graph in Figure 7a. If we color its vertices (fig. 7b), and isolate the vertices that received one color (fig. 7c), it is apparent that these vertices tend to be far apart. The reason for this result is simple: in a proper vertex coloring, adjacent vertices must receive different colors, and in a neighbor graph, adjacent vertices are nearby in space. Hence any two vertices with a given color cannot be nearby in space. This property holds for all three colors used (figs. 7d,e).
Figure 7. a) A neighbor graph, b) colored with 3 colors. Notice that c) points with the first color, d) the second color, and e) the third color all are farther apart than the original points.

Figure 8. a) A neighbor graph, colored with 3 colors. If b) the points with the first color are isolated, c) their neighbor graph is built and colored again, and d) points with the first color are isolated again, then these points are very far apart.

3.3. Hierarchical Coloring

This observation leads to an algorithm. We seek points that are as far apart as possible, and, as just discussed, graph coloring yields subsets of points that are somewhat far apart. To increase the separation further, coloring can be applied yet again.

This increased separation is shown in Figure 8. It shows the same neighbor graph again, 3-colored in the same way as before. The subset of points with color 1 is isolated, and its neighbor graph is built again. When that graph is colored in turn, the two points that receive color 1 are very far apart, in the original set of points.

Thus, by repeated colorings, we have found a way to identify individual pairs of points that are
very far apart. As discussed in the introduction, the problem we wanted to solve, is how to find a consecutive sequence of far-apart points. Such a sequence can be obtained, by carefully recording the colorings applied to each point. By recording these colorings, we obtain the final component of our algorithm.

The algorithm is simple: given a set of points, first build the neighbor graph, and color it. Some vertices will receive the same color. For each such color, isolate those vertices, and repeat the process: build their neighbor graph and color it again. The algorithm is described in more detail below, but is may be more easily understood through an example.

Look carefully at Figure 9. The top row (left) shows a set of points, and (right) the initial coloring of their neighbor graph. The next row below the top shows the subsets of points with color 1, 2, and 3, respectively, from left to right. For each subset, the neighbor graph has been obtained, and the points colored again. However, each point now has a two-digit label: the first digit is the color it received in the first coloring (top row), and the second digit is the color it received on its second coloring. In the left subset, two points were colored 1 twice (they have the label 11). In the third row, those two points are isolated again, and their graph is built and colored. Each of these two points gains another digit. By chance, the same thing happens to the middle and right subsets. The third row shows the result of this third coloring pass.

The process stops when each point has a different multi-digit label. The final labels appear on the bottom row, left. If the labels are now sorted lexicographically (i.e., they are treated as multi-character strings, not as numbers), we obtain the sequence (111, 112, 12, 13, 211, 212, 22, 23, 311, 312, 32, 33). This sequence contains the key result of the algorithm: if the points are visited in the order of this sequence (bottom row, center), then the distances between consecutive points will all tend to be relatively large.
Figure 9. Example of the HGC algorithm in action. Top row: an initial set of points, and their colored neighbor graph. Second row: for each color, a subset of points is extracted, and the graph for those points is built and colored again; a digit is appended to each label indicating the second coloring choice. Third row: the process is repeated again. Bottom row: the points with their final labels, and the visitation sequence obtained by sorting the labels. A slightly different visitation order is on the bottom right, with greater separation between points 112 and 12.
3.4. HGC algorithm

The steps followed in the example above can be formalized into an algorithm, which we call the hierarchical graph coloring (HGC) algorithm. Pseudocode for the HGC algorithm is shown on Figure 10a.

Given an input set of points $S$, the HGC algorithm proceeds as follows: first, the neighbor graph $G(S)$ is obtained from $S$. The vertices of $G(S)$ are the points in $S$, and the edges in $G(S)$ connect nearby points. The graph $G(S)$ is colored in some way, using $k$ colors ($k$ is not known a priori, and depends on the method used to color the graph).

For each of the $k$ colors used, let $S_j$ be the subset of points that received that color. The neighbor graph $G(S_j)$ is built in the same way that $G(S)$ was built, and this graph is colored in turn.

The process is repeated recursively. Each time a graph is colored, the subsets with a given color will be smaller, until subsets with a single point are obtained. At that point, recursion stops.

As the algorithm proceeds, a label is constructed for each point. Initially, every point has an empty label. When a point is colored with color $j$, the digit $j$ is appended to the point’s label. When the algorithm is finished, each point will end up with a multi-digit label.

In the final stage, the points are sorted, by comparing their labels lexicographically. In other words, they are sorted alphabetically (for example, 112 comes before 22, lexicographically). Equivalently, the labels could all be padded on the right with zeroes to make all the labels equally long, and then the labels sorted numerically.

When the points are visited in sorted order, the result will approximate the solution to the problem we set out to solve above in Section 3: consecutive points will tend to be distant in space, and even points two or three steps apart in the sequence will also be spatially distant.
Initialization:

for all points $p_i$ in $S$

$$l_i \leftarrow \text{empty label}$$

Labeling:

relabel($S$) {

if ($|S| > 1$)

construct $G(S)$

color_graph($G(S)$) with $k$ colors

for each color $j$ in the coloring

$S_j \leftarrow \text{subset of points with color } j$

reorder($S_j$)

rotate($S_j$) (optional)

append $j$ to all labels in $S_j$

relabel($S_j$)

}

Ordering:

sort the points in $S$

in increasing label order.

color_graph($G$)

Initialization:

$C \leftarrow \emptyset$

for all $v$ in $G$

$v$.color $\leftarrow$ empty

Coloring:

repeat $|G|$ times: {

$v \leftarrow \text{nextVertex}()$

$C_i \leftarrow \text{set of colors from all visited neighbors of } v$

if ($C_i = C$) then

$c \leftarrow \text{new color}$

add $c$ to $C$

else

$c \leftarrow \text{nextColor}(C - C_i)$

$v$.color $\leftarrow c$

}

Figure 10. Left: algorithm for graph-ordering a set of points $S$ Right: greedy vertex coloring algorithm for graph $G$
3.5. Subset Rotation

In the sorted sequence of points, consecutive points will usually be far apart, but sometimes we may be unlucky. For example, in Figure 8, the two points labeled 112 and 12 are consecutive, but are not substantially separated. What went wrong? The answer can be found by considering the first four points in the sequence: (111, 112, 12, 13). All these points were colored 1 on the first pass (they made up subset $S_1$). On the second pass, points 12 and 13 were colored 2 and 3, respectively. Had they been colored 3 and 2 instead, they would have received labels 13 and 12, respectively. The points would then be visited in the order shown at the bottom right of fig. 7. In other words, if a different (but still proper) vertex coloring is used, the separation between consecutive points may be increased.

Unfortunately, this improvement now causes points 13 and 211 to be close together. But all is not lost. Consider the sequence of points (111, 112, 12, 13, 211, 212, 22, 23), and break it up into two blocks: (111, 112, 12, 13) and (211, 212, 22, 23). After the first coloring pass, subset $S_1$ contained the points in the first block (the first four points), and subset $S_2$ the points in the second block (the next four). The points within each block are sequentially far apart, but at the boundary between blocks the separation is not great. To increase the inter-block separation, the points in the second subset could have been permuted (re-ordered) before their graph was colored. In fact, they could have been permuted such that the first point was as far as possible from the last point in the first subset. The purpose of the function $\text{rotate}$ is to perform just such a permutation.

The function $\text{rotate}$ examines a list of points $S_j$, and returns a cyclic permutation of it. In this cyclic permutation, the point in $S_j$ farthest last point in $S_{j-1}$ is placed at the first position in $S_j$. As long as the graph-coloring algorithm used is greedy, and always chooses color 1 for the first point in $S_j$, this will maximize inter-block separation, as was our goal. Since $\text{rotate}$ is invoked for
every subset (see the algorithm in fig. 10a), the result is increased separation between higher-level blocks too.

### 3.6. Graph Coloring Algorithm

In the HGC algorithm described above and shown in Figure 10a, coloring a graph is a key step, repeated many times. The original neighbor graph and also the graphs obtained recursively from point subsets must all be colored. However, we have not yet described a way to color a graph. In this section, we present a simple algorithm for coloring a graph.

Many graph coloring algorithms have been invented, and graph coloring continues to be actively researched. Proper vertex colorings that use fewer colors are usually desired, but for any given graph, finding a minimum coloring (one that uses the least number of colors) is known to be NP-hard. For this reason, most algorithms that are reasonably fast will not find a minimum coloring. In this work, we choose a simple, greedy algorithm, which unfortunately does not always produce a minimum coloring, but has the advantage of being fast and flexible. Our algorithm appears in Fig. 10b.

The greedy algorithm works by visiting every vertex of the graph once, and coloring it. It does not backtrack, but simply colors the vertex and continues to the next one. When a vertex is processed, some of its neighbors may already have been colored, and some may still remain to be processed. Simply by ensuring that the current vertex gets a color different from its already-colored neighbors, a proper vertex coloring is guaranteed to result. Remember that in a proper coloring each vertex’s color is constrained to be different from its neighbors’ colors.

Despite the proper-coloring constraint, there are two places where multiple choices are possible. First, the order in which the vertices are visited can be chosen in several ways: we consider list
order, random order, and breadth-first traversal. The `nextVertex` function deals with this problem, and is described below. Second, when choosing a color different from the available neighbor colors, there are many choices too: among possible valid colors, we consider choosing the first one, the least-frequently used, or a random color. This aspect is handled by the `nextColor` function, also described below.

The function `nextVertex` chooses a vertex among those not yet colored. Three variations were studied: 1) choose the next vertex in list order; 2) choose the next vertex randomly; 3) choose the vertex in breadth-first order. When the points are on a square grid, the first choice leads to dither matrices resembling Bayer’s dispersed dither. For such points on a grid, “list” order corresponds to a row-by-row visitation of the points. The second variation is meant to create multiple orderings of the same set of points, and leads to dither matrices that resemble random dither. The last choice is a heuristic that helps to reduce the total number of colors used, in many cases. Its effect is most evident for a graph with hexagonal symmetry (Fig. 16a). In such a graph, if nodes are visited in list order, at least 4 colors will be used, whereas a hexagonal graph can be colored using only 3 colors (it is 3-colorable).

The function `nextColor` chooses a color for the vertex whenever the algorithm has a choice among currently used colors not present among the current vertex’s neighbors. For example, if a vertex has four neighbors with colors \((1, 3, 4, 1)\), and the current set of colors used is \(C = \{1, 2, 3, 4, 5\}\), then the algorithm can choose to re-use colors 2 or 5, neither of which occurs among neighbors of the vertex. Again, three variations were studied for this choice: 1) choose the first color in the list; 2) choose the least-frequently used color; 3) choose a color randomly. The first choice also helps to create Bayer-like dither matrices. The second ensures that, for each color used, the number of vertices with that color is approximately the same. The last one, again, is
meant to create multiple orderings.

The function `reorder` provides yet another variation to the HGC algorithm: it permutes the subset. As argued above in §3.5, the points in a subset may be permuted without substantially affecting their mutual separation. Such a permutation creates variations in the graph order, which will yield many different threshold matrices. The function can either do nothing (the identity permutation), or apply a random shuffle of the points, which we call a *scramble*.

4. HGC VARIANTS

In the HGC algorithm, several choices can be made for some steps. The choices studied here involve the `next-node` function, the `next-color` function, the `rotate`, and the `reorder` function. In order to study the effect of these choices, they were combined in several ways, to obtain four major variants of the HGC algorithm.

The four variants studied are as follows: The “scrambled-subset” (SS) choice permutes subsets, and invokes “rotate”, but in the graph-coloring step selects nodes in list order and colors in least-frequently-used (LFU) order. The “non-rotated” (NR) choice is similar, but does not invoke “rotate”. The “random-node” (RN) choice selects nodes in random order, selects colors in LFU order, invokes rotate, but does not permute subsets. The fourth and last choice studied, “random-color” (RC), selects nodes in list order, colors in random order, invokes “rotate”, and does not permute subsets. These four choices are summarized in table 1.

Each of these four algorithms introduces a variation at one of the four crucial steps in the algorithm. The importance of these four steps will become evident in the quality of the orderings produced by the corresponding algorithm.

In each of the four variants, randomization is used at some point. Each time an algorithm runs,
<table>
<thead>
<tr>
<th>Name</th>
<th>next-node</th>
<th>next-color</th>
<th>rotate?</th>
<th>subset-scramble?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>list</td>
<td>LFU</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SS</td>
<td>list</td>
<td>LFU</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NR</td>
<td>list</td>
<td>LFU</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>RN</td>
<td>random</td>
<td>LFU</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>RC</td>
<td>list</td>
<td>random</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 1. Variants of hierarchical graph coloring algorithms investigated.

A different ordering is likely to be produced. Having many orderings for each algorithm allows us to study the results of the algorithm statistically. Once a good algorithm has been identified, having many orderings has an added benefit: it helps to avoid the repetitive artifacts that occur when a single dither matrix is used to halftone an image; instead of repeating the same matrix, different ones can be chosen at random.

5. SEPARATION STATISTICS

The stated goal of HGC algorithm was to maximize separation between consecutive points. In this section we show that the separation between consecutive points is indeed large.

As an experiment, a set of 1024 points is considered, arranged in a $32 \times 32$ regular square grid. The grid spans a $1 \times 1$ region of the plane, so each point’s $x$ and $y$ ranges from 0.0 to 1.0. One run of the HGC algorithm is applied, using the SS variant, yielding an ordered sequence of points. The distance between consecutive points in the sequence is considered.

Figure 11a shows all the separations in the sequence, with sequence number on the $x$ axis and separation on $y$. Each data point on the graph shows the separation between one consecutive pair points in the sequence. Notice that almost all separations are $\geq 0.25$ (only five data points have
27 2.5). Most separations are from 0.35 to 0.65. For comparison, fig. 11b shows the separation when the points are ordered randomly. Clearly the HGC algorithm does increase consecutive-point separation, as was our goal.

These scatter plots are suggestive, but a better measure of the quality of the ordering is obtained by counting the frequencies of the separations. This is equivalent to projecting the points in the scatter plot onto the y axis, and obtaining histograms of the resulting sample densities.

Figure 12 shows these counts, for the four algorithms. To reduce noise, the counts show the averages over 100 runs of each HGC variant. The plot shows that the SS approach maximizes the separation between consecutive pixels, followed by the RC algorithm. The next best are the RN and NR algorithms. By comparison, the worst case occurs when nodes are colored in random order.

Notice that the counts all rise as separation increases, then fall down to zero for higher separations. The reason for this fall is that separations must be measured toroidally, since the points lie in a repetitively-tiled dither matrix. Toroidal adjacency is discussed below in section 5.3.

5.1. Converse of Hypothesis

At the outset of this paper, we posited this hypothesis: enforcing a large separation between every pair of consecutive thresholds will produce good-quality dispersed-dot dither matrices. Here, we must emphasize that the converse of the hypothesis does not hold. One counterexample is Ulichney’s void-and-cluster algorithm.11 This algorithm is widely recognized as yielding good dither matrices. However, the consecutive-separation scatter plot for such a matrix is very similar to the random-threshold plot in fig. 11b (and hence this scatter plot is not shown). In other words, there are good dispersed-dot dither matrices without large consecutive-threshold separations. Despite
Figure 11. Separation between consecutive points graph order: a) SS algorithm; b) random order. Both orderings use a 32×32 raster of pixels inside a 1×1 square.

this fact, the HGC algorithm is worth examining, because it can produce a variety of threshold matrices, and can studying it yields insights on dispersed-dot dithering in general.

5.2. k-Colorability and Good Gray Levels

As we stated in section 3.1, some graphs can be colored with fewer colors than others. It turns out that the minimum number of colors needed for a graph plays an important role in the quality of the halftone patterns that the HGC algorithm produces. It also guarantees that some gray levels (some shades of gray) are special: for some variants of the HGC algorithm, those shades are halftoned better, with fewer visible clumping artifacts. In this section, we will explain why this happens.

Figure 18 shows a “gray ramp” (a gradually darker rectangle), halftoned with the RC variant. It is apparent that some gray levels exhibit “cleaner” halftone patterns. Why is this so? The answer is related to the number of colors required to color a graph’s nodes.

Many different dither matrices are used in fig. 18, but they all contain a 32×32 square grid of
points. For this set of points, the neighbor graph is the 4-neighbor raster graph (shown in fig. 6a). This graph is 2-colorable. Because the RC variant visits the points in list order (row by row), the very first coloring of these points will actually yield the 2-color coloring shown in the figure. The interested reader can easily verify this fact, by manually coloring the vertices of such a graph row by row.

After the first coloring, the HGC algorithm divides the points into two subsets $S_1$ and $S_2$, one for each color. There are 1024 points altogether, and by symmetry (see the figure), both subsets have equal sizes (both contain 512 points). All 512 points in $S_1$ get an initial label 1, and all 512 points in $S_2$ get labeled 2. Subsequent labelings append digits to the labels, but the first digit remains unchanged. When the labels are finally sorted, the points in $S_1$ will all come before points in $S_2$, because for all of them, the first digit is smaller.
This simple fact can be used to predict the halftone pattern for an image with gray level 0.5. Look at the points colored 1 in fig. 6a). They make up a checkerboard pattern. Recall from section 3 that, for the $i^{th}$ point in the sequence, the threshold in the dither matrix is $i/(mn)$. The first 512 points in the sequence yield thresholds up to $511/1024$ (i.e., $< 0.5$). Thus all entries in the dither matrix with threshold $< 0.5$ occur in a checkerboard pattern. Hence an image region with gray level 0.5 will be halftoned with a checkerboard pattern of black and white pixels.

Such a checkerboard pattern is the most desirable for this gray level, because it is a square grid rotated $45^\circ$, and the human visual system is least sensitive to patterns aligned at $45^\circ$.

The argument can be extended to other gray levels. Since the points in $S_1$ occupy a square grid (although it is rotated $45^\circ$), their neighbor graph will be a 4-neighbor graph again. The fact that the graph is rotated does not change its structure. It is still 2-colorable, as seen in fig. 6d. Thus, when the 512 points in $S_1$ are colored the second time, two colors will be used, and the set will be divided into two equal-size subsets $S_{11}$ and $S_{12}$, with 256 points each. Fig. 6d shows that points in $S_{11}$ make up a square grid, but this grid is spaced twice as wide as the original grid of 1024 points. By an argument similar to the one just given, this implies that the thresholds with values $< 0.25$ must occupy that square grid. Hence an image region with gray level 0.25 will be halftoned with a regular grid of white pixels on a black background.

Similar reasoning tells us that gray levels 0.125, 0.0625, etc. will be halftoned with a rectangular grid or a checkerboard of white pixels on a black background. The argument can also be used to predict that gray levels 0.75, 0.875, 0.9325, etc. will be halftoned with a rectangular or checkerboard of black pixels on a white background.

Now, consider a hexagonal array of points. Its neighbor graph is 3-colorable, as seen in fig. 16c. Following the same line of reasoning as above, we expect good halftone patterns at gray levels $1/3$
Figure 13. Gray test ramp for hexagonal dither

and 2/3, because the first coloring pass will split the points into three equal subsets $S_1$, $S_2$, and $S_3$. Moreover each of these three subsets consists of a hexagonal array of points too, albeit more widely separated than the original array. Hence, applying the same argument again, we expect good halftoning patterns for gray levels $1/9$, $1/27$, etc. and $8/9$, $26/27$, etc.

Furthermore, since a hexagonal graph cannot be 2-colored, we expect gray level 0.5 to exhibit a bad halftone pattern for a hexagonal grid: points tend to cluster into noticeable artifacts. Fig. 13 confirms our predictions: pixel values 226 and 170 (corresponding to gray levels $8/9$ and $2/3$) show regular hexagonal patterns, whereas pixel value 128 (gray level 0.5) shows clusters. Further evidence of this can be seen in Fig. 14, which shows counts of inter-point separations of the halftoning patterns for gray levels $2/3$ and $1/2$. The point-separation distribution for gray level $2/3$ shows perfect hexagonal symmetry, whereas for gray level $1/2$ the symmetries involved are more complicated, which reflects the small clustering artifacts in the halftone pattern.

As a consequence of this discussion, it becomes clear that, when halftoning with a hexagonal
grid, no halftoning scheme can possibly produce an artifact-free 1/2 gray level. If a well-separated halftoning pattern for 50% gray existed, it would imply that two colors on a hexagonal graph would suffice to color nodes such that adjacent nodes were colored differently; in other words, a hexagonal graph would be 2-colorable, which is not the case. This limitation does not apply just to our results, but is inherent to hexagonal halftoning in general.

Finally, consider an arbitrary collection of randomly-placed points. Its neighbor graph will be planar, and thus is 4-colorable, as the 4-color theorem states. If the greedy coloring algorithm used here is able to find such a 4-coloring for this collection of points, then we expect that good, cluster-free halftone patterns will occur for gray levels \(k/4\), \(k = 1, 2, 3\). However, a test gray ramp in Fig. 23 shows that this effect, if present, is drowned out by the randomness of the points being used for halftoning.

### 5.3. Tiling Effects

When a very large image needs to be halftoned, a dither matrix of size equal to the image could be used, but this is rarely done. Instead, a much smaller dither matrix is created once, and then repeated across the image, just like a regular array of square floor tiles. There are two reasons for this choice: first, less storage space is needed for a smaller matrix, which is good for simple...
printing devices where halftoning is usually done, devices which may have limited memory. Second,
and more importantly, building a dither matrix in the first place can be very time-consuming, so
creating a fresh large matrix for each image destroys the simplicity and speed that make dithering
attractive in the first place.

Because a dither matrix will usually be repeated in a “tiled” pattern, the corresponding neighbor
graph for the matrix must take the repetition into account. Consider Figure 15b, which shows
a 3x3 matrix repeated nine times. The neighbor graph for the large 9x9 matrix could be built, and
then the HGC algorithm used to obtain a dither matrix. However, the same effect can be obtained
by using the original small 3x3 matrix, and adding extra edges to its graph.

For example, point a has neighbors b and d in the 3x3 graph (Figure 15a), but after repetition
it acquires new neighbors, c and h. Figure 15c shows the same graph as fig. 15a, but with new
edges (a,c) and (a,h) added to reflect the additional neighbor relationships induced by repetition.
Similar edges are added to connect points on the left edge of the block with points on the right
dge, and points on the top edge with the bottom edge. We call this augmented graph the \textit{toroidal
neighbor graph}. The word “toroidal” is used because building the new graph is similar to building
a torus from an elastic square: first curve the square into a tube, and join together what were once
opposite edges. Then, curve the tube into a torus, and join what was the top circular end of the
tube with the bottom circle.

With these extra edges in place, the HGC algorithm produces better results, and avoids obvious
clumps of pixels at the “seams” where the copied dither matrices meet. Why? Suppose the original,
non-toroidal neighbor graph were used. If so, then on the very first coloring pass of the algorithm,
it would be possible for a point on one edge (for example a) to be colored the same as a point on
the opposite edge (c). If this occurred, then both points would be part of the same point subset \(S_j\),
Figure 15. Toroidal wrap-around: a) $3 \times 3$ square grid of points and b) the same grid repeated across an image. c) the same grid, with wrap-around edges added to represent the neighbor relationships implied in the repeated grid.

and their labels would have the same first digit $j$. Hence they would be lexicographically similar, and after sorting, would be relatively close in the sorted sequence. In other words, the threshold values assigned to the two points would be close to each other. If the resulting dither matrix were applied to a region of constant gray value just below the threshold values of those two points, pixels at the seam would then both be set black, and a small clump of two pixels at the seam would be produced. On the other hand, if the extra edges are in place, points on the “seams” must get different colors, and hence will be further apart lexicographically, hence will get threshold values farther apart, hence will be less likely to both be set black, and hence there will be fewer clumps at the seams.

Tiling With a Hexagonal Grid : The above discussion assumes that the pixels lie on a regular square grid. However, pixels may be arranged on a hexagonal grid instead. Such an arrangement packs pixels slightly tighter than a square grid does, and happens to resemble the arrangement of
For pixels on a hexagonal grid, should a rectangular block of pixels be used as a dither “matrix”, to be repeated across an image just as for pixels on a square grid? No. Consider Figure 16a, which shows just such a rectangular block of pixels, even though the pixels are arranged in a hexagonal pattern. Adding extra edges to account for toroidal adjacency leads to a graph that requires four colors. Notice that colors 1, 2, and 3 are used approximately equally often, but color 4 is used left often, only on two points. Moreover, the points colored 4 are all on one side of the block. When this first coloring is used, the unequal distribution of colors will lead to an obvious artifact on one edge of the repeating block. Figure 17 shows what happens when a larger rectangular block of hexagonal-grid pixels is used to halftone a gray test ramp. The obvious horizontal white line occurs because, in this case, only a few points in the block (near the top of the block) get color 4.

The proper way to repeat a block of hexagonal-grid pixels is to not use a square block. Instead, a hexagonal block of pixels should be used (see Figure 16c). Such a hexagonal block cannot be replicated using a square tiling pattern, without leaving gaps. Instead, the block must be repeated in a hexagonal pattern itself, much like the cells in a honeycomb (see Figure 16b). Such a repeating pattern has adjacencies between edge pixels on opposite sides of the block, just as for a square grid of pixels. Adding edges to account for these adjacencies leads to a graph that is 3-colorable, and this greatly reduces visible artifacts. Although these extra edges are not shown in Fig. 16c because they would greatly clutter the illustration, the reader should consider tiling this colored block in the way shown by Fig. 16b to persuade him or herself that the resulting graph is indeed properly colored.
Figure 16. Hexagonal wrap-around: a) A rectangular block of points on a hexagonal grid may require four colors if toroidal symmetry is used. b) If a hexagonal block of hexagonal-grid points is tiled, using a hexagonal tiling pattern, c) the resulting graph is 3-colorable (wrap-around edges are not shown).

Figure 17. Halftone pattern obtained by ordering a set of points on a hexagonal grid. The points are contained in a rectangular region, which is tiled with toroidal symmetry. Notice the strong horizontal artifacts, caused by the inappropriate use of toroidal symmetry.
6. RESULTS

6.1. Test Images and Statistics

Figures 18, 19, and 20 show gray ramps for the RC, RN, and SS algorithms, respectively (the results for the NR algorithm are very similar to the SS algorithm and are not shown here). A single threshold matrix from each algorithm could have been replicated across the ramp, but this would have introduced distracting repetitive artifacts. Such artifacts are evident when Bayer’s dither matrix is used. To avoid such repetition, we can use many different matrices of the same type. It is easy to obtain variants of the same type of matrix. Recall that all three algorithms involve at least one randomization step, so each time an algorithm runs, a different threshold matrix is produced.

For each algorithm, 100 different 32×32 threshold matrices were constructed. These matrices were tiled across the ramp, but a different one of the 100 was chosen at random for each tile.

The RC algorithm produces dither matrices that most resemble Bayer’s. Because it visits the nodes in list (row-wise) order, and the 4-neighbor raster graph is 2-colorable, the nextColor function almost always has only one unused color available to it. Hence, usually no real random choice is made, and the algorithm runs almost deterministically, yielding the same colorings, and nearly identical dither matrices.

In contrast, the RN algorithm produces dither matrices that are much more random than the RC algorithm. In the RN algorithm, the next vertex to be colored is chosen at random, and not in row-wise order. As a result, it may not be adjacent to an already-colored vertex. For such an isolated vertex, a color choice is made which is unrelated to previously-colored vertices. As the greedy coloring progresses, more vertices are colored, and eventually a vertex will be processed.
which has two already-colored neighbors. However, due to randomness, these two neighbors will likely have different colors. Hence, the vertex being processed will require a third color.

Thus the resulting graph colorings will usually use 3 or more colors, not the optimal 2 colors which a 4-neighbor raster graph admits. This means each color will not comprise the checkerboard pattern appearing in Fig. 6a, and the resulting dither matrices will not exhibit the regular symmetric structure of the RC algorithm’s matrices.
The SS algorithm produces matrices which appear more random than the RC and less random than the RN matrices. In the SS algorithm, the vertices in each color subset are randomly ordered before being fed to the greedy coloring algorithm. In effect, the SS algorithm resembles the RN algorithm in that the vertices in each subset are processed in random order. However, this randomness does not apply the very first time the algorithm is invoked, when it must order the whole 32×32 raster graph. In this first pass, the 4-neighbor raster graph will be 2-colored. As a result, the halftone pattern for gray level 1/2 will be an artifact-free checkerboard, for reasons explained in §5.2. For other gray levels, however, the random node order that the subset scrambling produces becomes more pronounced. Thus, for very light gray levels the halftone pattern resembles the RN algorithm’s.

A hybrid of the SS and RC algorithms appears in Fig. 21. Here, the subset-scrambling step is not invoked every time. Instead, the depth of recursive calls to the relabel function is taken into account, and subsets are only scrambled for recursion depth greater than 4. In effect, the first four levels of graph coloring resemble the RC algorithm, and the “good” gray levels 1/2, 3/4, and 7/8
Figure 21. Gray test ramp for SS algorithm, but subset scrambling is only implemented for recursion depth above 4.

are now preserved, and randomness appears only elsewhere in the ramp.

Figure 22 shows the point spread distributions (PSD, or autocorrelation function) of the halftone patterns at 15/16, 13/16, 3/4, and 1/2 gray levels, for the three (SS, RC, RN) graph-order algorithms. The plots are experimentally derived from a typical run of the corresponding algorithm. In each plot, a circle’s position indicates a displacement from the current pixel, which is at the center of the plot. The circle’s area is proportional to the number of times that displacement occurred. The center (zero displacement) pixel itself is omitted, as it would otherwise overwhelm the other displacements.

These plots illustrate the behavior described in the preceding paragraphs: for the SS and RC algorithms, the 1/2 gray level is a perfect checkerboard, and the PSD reflects this structure. However, the RN algorithm’s 1/2 gray level is rather random and exhibits no such checkerboard, which is indeed absent in its PSD plot. For very light gray levels, on the other hand, both the SS and RN algorithms exhibit randomness, and this in turn is reflected in their PSDs. Finally, the
Figure 22. Point-spread distributions for three graph-ordering algorithms. Top row: SS algorithm. Middle row: RC algorithm. Bottom row: RN algorithm.

RC halftone, much like Bayer’s halftone, shows square or checkboard symmetry for a range of gray levels.

6.2. Random-Point Halftoning

So far, the halftoning methods described have used points that initially lie on a rectangular grid. However, the method was designed to work with points in any position. As an example, consider points which are initially placed randomly inside a $1 \times 1$ square, and ordered. The resulting halftone pattern obtained (with the RC algorithm) is shown in Fig. 23. When rendering the test ramps shown, each point is replaced by their corresponding region in the points’ voronoi diagram. In this
particular illustration, small gaps are introduced between the Voronoi regions to make the regions more evident.

6.3. Non-$2^k$ Dither Matrices

The generality of the HGC algorithm presents us with an opportunity to study various sizes of dither matrices. Figure 24 shows gray ramps dithered with matrices of sizes $3 \times 3$ through $16 \times 16$. The standard version of the HGC algorithm was used to generate all matrices; no randomization was used. Not surprisingly, larger matrices produce better results. More interestingly, odd-sized matrices produce halftone patterns with more clustering artifacts; even-sized matrices are better.

The most likely reason for this result is that an odd-size array of points, toroidally wrapped, has a 4-neighbor neighbor graph which is not 2-colorable. For example, the graph shown in Figure 15c is not 2-colorable; it needs at least 3 colors. Moreover, the first coloring for an odd-sized matrix will usually not divide the points into subsets with equal numbers of points. Worse, the points in each subset will not usually be symmetrically arranged, but rather exhibit some clustering, and this clustering shows up in the final graph ordering.
Figure 24. Gray test ramp for dither matrices of various sizes.
6.4. Comparison of images

Test ramps are useful, but to properly test a halftoning algorithm it must be applied to real images, because the final user of a halftoned image is a human being. Figure 25 shows most of the algorithms presented above, applied to a photograph. For comparison, the original image is shown on the top row, along with halftones using Bayer ordered dither and Floyd-Steinberg error diffusion.

The second row shows the RN, RC, and SS algorithms applied to the image. The RN algorithm produces the noisiest results, although not quite as severe as true random thresholding. The RC image is similar to the one above it (Bayer’s ordered dither), but shows less of the regular structures evident in the upper image, thanks to the slight randomness in the matrices used. The SS image on the right shows more randomness than the RC image. Just like the RC image, it uses an innocuous checkerboard pattern at gray level 0.5, which may be considered an improvement on the same regions in the error-diffusion image above it.

The third row shows some of the additional methods studied. On the left, the halftone uses a 12×12 dither matrix, showing some obvious repetitive artifacts (a single matrix was used, tiled repetitively). The middle shows hexagonal dither. Notice the slight clustering artifacts in those same 0.5-gray regions where the RC and SS algorithms use a more pleasing checkerboard. This visually supports the argument presented above in section 5.2, proving that no hexagonal dither can avoid such artifacts. Finally, the image on the right shows a random-point halftone, with each point drawn as a small voronoi polygon.

6.5. Artistic Halftoning

The graph-ordering algorithms presented above all strive to make the dither pattern itself as inconspicuous as possible. However, there are situations when the dither pattern should stand out.
Figure 25. Comparison of halftoning methods: a) Original image; b) Bayer dither; c) Floyd-Steinberg error diffusion; d) Random-node algorithm; e) Random-color algorithm; f) Scrambled-subset algorithm; g) 12x12 ordered dither; h) Hexagonal dither; i) Random-point dither.
If a repeated dither pattern is made to stand out, it can produce an artistic effect.

This effect has been called *artistic halftoning* by Ostromoukhov.\(^1\) It uses two images: an overall image to be reproduced, and a much smaller *motif* image, which is replicated across the overall image. When halftoning is applied, the overall image is perceived, but the repeated motif images are also evident (Figure 26a shows an example).

The key problem in artistic halftoning is how to obtain a dither matrix from the motif. The motif should be a gray scale image, so full-color motifs should be turned to gray scale first. Once this is done, the main requirement in the dither matrix is that bright pixels in the motif become high thresholds in the matrix, and dark pixels become low thresholds. As a result, when this matrix is applied to a gray-scale image, a bright pixel in the image is likely to be set white in the halftoned result, and a dark pixel is likely to be set black. If the dither matrix is applied to a full-color image instead, the rules for choosing the final color at each pixel are slightly more complex, but they have been described elsewhere,\(^1\) so we do not dwell on them here. Whatever the source image, the repeated motif in the result will be reproduced clearly, although modulated by the original image.

Another important problem in artistic halftoning is how to handle motif images that contain very few gray levels. In the most extreme case, a motif image may contain only two levels: black and white. In such a case, we cannot simply replace black motif pixels with a fixed low threshold (*e.g.* 0.25), and white pixels with a fixed high threshold (*e.g.* 0.75): doing so tends to produce large regions of black and white in the resulting halftoned image, and most of the intermediate tones in the original image will be lost.

To retain intermediate tones in the result, the motif image must be altered, by introducing more gray levels, in such a way that original motif is still discernible. Ostromoukhov addresses
this problem by reducing the motif to a set of skeleton curves. Then the threshold at a given pixel is set to the relative distance from the pixel to the nearest curve. Thus thresholds fall off smoothly away from the curves in the motif. The resulting halftone contains curves of varying thickness, thicker curves reproducing darker tones in the source image, and thinner curves lighter tones. Unfortunately, in very dark regions, the resulting curves appear very thick and may merge into each other, so that the shape of the curves is lost. Ostromoukhov’s solution is to replace the highest thresholds with random values, which preserves some gray values using random thresholds, but unfortunately adds noise to the halftone result. Figure 26a shows an artistic halftone that uses a spiral motif, produced with Ostromoukhov’s approach.

The HGC algorithm can be adapted to yield an artistic halftoning method which is more flexible than Ostromoukhov’s. Any motif can be used, including motifs with very few gray levels, and there is no need to simplify the motif into skeleton curves. The process is straightforward: apply HGC to each gray level separately. To be more precise, the following algorithm is applied:

1. Separate the motif image into subsets of pixels, one subset for each gray level in the image. For each gray level $g$, let the point set $S_g$ contain all the pixels with that gray level.

2. Apply the HGC algorithm to each $S_g$, obtaining an ordered list of points $O_g$.

3. Then, concatenate all the lists $O_g$ into a master list $O$, containing all the pixels in the motif.

4. Finally, create a dither matrix by setting each threshold according its position in the list $O$. In other words, the threshold at $(xy)$ should be $t = i/|O|$, where $i$ is the index of point $(xy)$ in the sorted list $O$.

This adaptation is most effective for binary (black-and-white) motifs such as text. In such a motif, all the black pixels are ordered into one list $O_0$, and the white pixels into another list $O_1$. Within
each list, HGC produces a separate halftone pattern, which is capable of reproducing a whole range of tones in the original image, using dispersed-dot dither. The concatenated list $O$ is still a dispersed-dot dither, but the pixels in the black part of the motif have higher thresholds than those in the white region. As a result, in a halftoned image, darker colors will tend to occur in the black parts of the motif, and brighter colors in the white parts. Thus the motif will show through, as desired in artistic halftoning.

Figure 26b shows the result of applying this method to the same image as fig. 26a, using a thicker spiral motif. Notice that medium-brightness colors are retained faithfully. The motif is less stark, and more details in the original image can be seen.

We need not use a single motif in artistic halftoning. We can use each letter of the alphabet as a different motif. By following the procedure described above, the HGC method can turn each letter into a dither matrix. This lets us superimpose text on an image. Figure 27 shows how multiple black-and-white motif images (the 26 letters of the English alphabet) may be used as motifs to illustrate the text which the halftoned image is referring to.

It should be emphasized that this improvement on artistic halftoning is new. Its effectiveness is a result of increasing the number of different thresholds in the dither matrix. With more thresholds, more shades can be obtained in the resulting halftone. The HGC algorithm is an integral part of the improvement: without it, it's difficult to ensure that the thresholds each subset $S_g$ occur in a dispersed-dot pattern. The locations in $S_g$ can be arbitrary, and won't usually occur in a square grid. Hence a method that yields dispersed-dot dither for arbitrary locations is needed.

For binary motifs, this improvement preserves the shapes of foreground objects, and renders these objects in a halftone pattern, as can be seen in the detail on fig. 26b. In contrast, in Ostromoukhov's approach the shapes in the motif must be thickened or thinned to yield darker or
Figure 26. a) Artistic dither with spiral motif, using Ostromoukhov’s method (with detail of nose); b) Using the HGC method (with detail).

lighter output tones. This varying thickness distorts the motif, and may make it unrecognizable.

7. CONCLUSION

At the beginning of this paper (section 1.1), we hypothesized that enforcing large separations between consecutive thresholds leads to good dispersed-dot threshold matrices. To this end, we derived an algorithm for creating such large consecutive separations.

The results obtained seem to support the hypothesis, in several ways. First, the algorithm yields Bayer’s dispersed-dot threshold matrices as a special case. Second it yields other Bayer-like matrices whose width and height are not $2^k$. Third, it yields dispersed-dot matrices for pixels on a
Figure 27. Graph-order dither used for artistic halftoning: a) Input image b) 26 binary motifs c) artistic screen, with text superimposed on image.

hexagonal grid. Fourth, the algorithm can be randomized in several ways, to construct many similar matrices, helping to avoid reperitive artifacts. Depending on where in the algorithm randomization is introduced, the effect on the resulting halftones can vary, ranging from Bayer’s highly structured pattern to matrices that resemble random thresholding. Fifth, the algorithm does not explicitly require that pixels lie on a square grid, so threshold “matrices” can be defined for pixels in random or other arbitrary positions. Finally, our solution also improves artistic halftoning, by extending the range of viable motif images that may be repeated when reproducing an overall image.

On the other hand, as pointed out in section 5.1, good threshold matrices exist in which the separation between thresholds is not related to consecutive threshold values. Thus our initial hypothesis needs to be refined: more work is needed.
Furthermore, many of the results obtained here (e.g., hexagonal-grid dither) have been achieved with other techniques. The advantage of the HGC algorithm is its flexibility: many different kinds of dither matrices can be obtained by varying the inputs and internal controls of the algorithm. Moreover, the HGC algorithm may lead to further improvements to dithering.

7.1. Future Work

The work presented suggests many avenues for future investigation. The graph coloring algorithm studied here is a simple greedy one, and it would be useful to study more sophisticated coloring methods. In particular, the present method ignores the relative orientation of a node’s neighbor when color choices are made; taking this orientation into account may yield better dither matrices. In addition, some of the halftones shown here exhibit anisotropic dot patterns, just like Bayer’s halftones. Hence another useful goal would be graph coloring algorithms for which each color subset contains points that approximate a blue-noise pattern. Such patterns are known to produce the best halftones. Yet another useful line of investigation may consider graph-ordering techniques in the parallel-computing literature, some of which use space-filling curves and others hierarchical methods. These may be adapted to halftoning as well. Finally, since the method uses graphs, it is not bound to two dimensions, and may be useful for creating higher-dimensional sampling patterns.

8. ACKNOWLEDGEMENTS

This paper greatly elaborates on ideas first presented in a conference publication. The author is very much indebted to Leslie Trueman, and to the anonymous reviewers, whose careful work and incisive comments contributed greatly to the substance and presentation of these ideas. Thank you.


