Halftone Postprocessing for Improved Rendition of Highlights and Shadows

C. Brian Atkins[†,*], Jan P. Allebach[‡], and Charles A. Bouman[‡]

[†]Imaging Technology Department
Hewlett-Packard Company, Hewlett-Packard Laboratories
Palo Alto, CA 94304-1126
cbatkins@hpl.hp.com

[‡]Electronic Imaging Systems Laboratory
Purdue University, School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285
{allebach,bouman}@ecn.purdue.edu

December 7, 1998

Abstract

Many binary halftoning algorithms tend to render extreme tones (*i.e.,* very light or very dark tones) with objectionable dot distributions. To alleviate this artifact, we introduce a halftone postprocessing algorithm called the *Springs* algorithm. The objective of *Springs* is to rearrange minority pixels in affected regions for a smoother, more attractive rendition. In this paper, we describe the *Springs* algorithm, and we show results which demonstrate its effectiveness.

The heart of this algorithm is a simple dot-rearrangement heuristic which results in a more isotropic dot distribution. The approach is to treat any well-isolated dot as if it were connected to neighboring dots by springs, and to move it to a location where the energy in the springs is a minimum. Applied to the whole image, this could degrade halftone appearance. However, *Springs* only moves dots in selected regions of the image. Pixels that are not minority pixels are not moved at all. Moreover, dot rearrangement is disabled on and around detected edges, since it could otherwise render those edges soft and diffuse.

[*]This work was performed while C. B. Atkins was with Purdue University.

1 Introduction

Digital halftoning is the process of transforming a "continuous-tone" digital image into one which may be rendered by a device having a relatively limited palette [1]. In the case of binary halftoning, the intended output device can only produce two colors — usually black and white. This case is especially important because of its relevance in the field of printing.

One drawback of many common binary halftoning algorithms is that they often render extreme tones (*i*.e., very light and very dark tones) with objectionable dot distributions. One such algorithm is error diffusion [2], which is known to produce a strongly directional "worming" artifact. Another example is screening using traditional dispersed-dot dither arrays (see [3], for example), which produce artificially periodic patterns.

What makes these behaviors so objectionable is that they suggest the presence of texture in regions where there should not be any texture. When it comes to printing binary halftone images, this is especially problematic in the case of light tones, where the minority pixels — black dots against the white background [4] — tend to be more visible. This is generally less of a problem in the dark tones, since the visibility of individual minority pixels — white dots against the black background — can be significantly reduced because of factors collectively referred to as "dot gain."

A variety of novel halftoning methods have addressed this problem. Included among them are methods for modified error diffusion [5], dither array generation [6, 7, 8], two-dimensional pulse-density modulation [9], and halftoning based on tree coding [10], which specifically address the problem of how minority pixels are arranged in regions of extreme tone. These techniques work by controlling the spacing among minority pixels in order to avoid the artifacts described above. Despite the satisfactory performance of many of these methods, error diffusion (as it was originally proposed) and traditional dither arrays are still in wide use.

In this paper we introduce a halftone postprocessing technique called the *Springs* algorithm, which has been developed to suppress the above-described artifacts. The objective of *Springs* is to rearrange minority pixels in affected regions, in order to give a smoother, more attractive rendition.

An important aspect of *Springs* is that it is separate and apart from whatever halftoning algorithm was used to generate the input halftone image. Moreover, *Springs* works without any knowledge of the continuous-tone image from which the input halftone was generated. This means that *Springs* is a halftone postprocessing algorithm strictly, and it distinguishes *Springs* from halftoning algorithms (or halftone optimization algorithms) that effectively perform halftone postprocessing such as [11], [12], [13], and [14]. However, *Springs* does share some similarity with the algorithm of Eshbach and Hauck described in [11], in that they are both based on treating minority pixels as if they were connected to neighboring minority pixels by imaginary springs.

To our knowledge, there is very little prior published work in the area of halftone postprocessing for improved rendition of extreme tones. One exception is [15], where we decribe an algorithm which is very similar to the one described here. Other work in the area of halftone postprocessing relates to the important but different field of digital document restoration [16, 17].

The remainder of this paper is organized as follows. In the next section, we describe the general structure of the *Springs* algorithm. Then, in Sec. 3, we give a detailed description of the dot-rearrangement procedure which forms the basis of *Springs*. In Sec. 4, we describe the scheme that is used to detect edge regions in the halftone, where dot rearrangement is not allowed. Finally, in Sec. 5, we show some experimental results; and in Sec. 6, we give some conclusions.

2 The *Springs* algorithm

The *Springs* algorithm works by first identifying regions where postprocessing may be warranted, and then rearranging minority pixels in those regions for a more homogeneous distribution. The two stages of this process are depicted in Fig. 1.

Note that this involves moving black dots in light regions, and it also involves moving white dots in dark regions. In this paper, we will use the term "dot" to refer to a minority pixel in a generic sense. There is little room for confusion here, since in regions of extreme tone, the meaning of the term *minority pixel* is unambiguous; and we will see in Sec. 3 that we do not even move pixels that are not minority pixels.

The movement of a dot amounts to the minimization of a cost functional. This cost functional represents the energy stored in virtual springs that connect the dot to some neighboring dots, as depicted in Fig 2. To minimize this functional, we allow the central dot to move, while the neighboring dots remain fixed. Intuitively this minimization has the effect of relaxing the dot distribution, which results in a more homogeneous texture. In order to evaluate the functional, we assume that the "rest length" of each of the springs is equal to the mean of the distances between the dot (at its starting location) and its neighbors. This means that the form of the functional actually adapts to the initial dot distribution.

Prior to the dot-rearrangement stage, *Springs* generates a binary map identifying edge regions, where dot rearrangment will not be allowed. This is important since subjecting edges to the dot-rearrangment process could render them soft and diffuse.

3 Dot rearrangement

The dot-rearrangement stage of the *Springs* algorithm is iterative. Each iteration consists of one raster-ordered pass through the lattice of pixel locations in the image. In the turn of each pixel location, we consider the pixel there as a candidate for being moved; and we move it, provided that certain conditions are met. These conditions are listed in the following paragraph. The form of the cost functional depends upon the position of the dot relative to the positions of N neighboring dots. An important aspect of the algorithm is the procedure by which we choose those neighbors.

In order for a pixel to be moved, it must satisfy three criteria. First, it must be spatially separated from any edges detected in the halftone. These regions are identified by a binary edge map. (We will describe how the edge map is generated in the next section.) Second, the pixel must not be adjacent to any other pixel of the same color. This is important for the preservation of delicate but continuous structures like thin lines and serifs in text. Finally, the mean of the distances between the pixel and its N neighboring pixels of the same color must be greater than some threshold value. For this threshold, we use a value of 3 pixel units. This is a requirement since the rearrangement process does not improve distributions that are too densely packed. Note that these last two rules effectively ensure that we will not move pixels that are not minority pixels.

3.1 Cost functional minimization

The dot movement is implemented as a sequence of short traversals which terminates at a local minimum of the cost functional. In the $(k+1)$–th traversal, the dot is moved from its location $n^{(k)}$ to the adjacent pixel location $n^{(k+1)}$ which decreases the cost the most. Any location which has no adjacent locations that would decrease the cost is a local minimum. The only restriction to this process is that the dot may not be moved into a location having another dot as a neighbor. We describe our motivation for this restriction below.

To evaluate the cost functional, it is only necessary to add up the energy stored in imaginary springs which connect the dot to its neighbors. If the dot is at location n, then the total energy $e(n)$ is

$$e(n) = \sum_{i=1}^{N} e_i(n) \ , \tag{1}$$

where $e_i(n)$ is the energy in the spring connected to the i–th neighboring dot. This is computed by squaring the magnitude of its displacement. Specifically,

$$e_i(n) = |d_i(n) - r|^2 \ , \tag{2}$$

where $d_i(n)$ is the distance between n and the location n_i of the i–th dot,

$$d_i(n) = \|n - n_i\| \ , \tag{3}$$

and r is the rest length of the spring.

We assume that the rest length r is the same for each of the springs. It is computed as the mean of the distances between the initial dot location and the neighboring dots; specifically, r is computed as

$$r = \frac{1}{N} \sum_{i=1}^{N} \|n^{(0)} - n_i\| \ , \tag{4}$$

where $n^{(0)}$ is the initial dot location.

Figure 3 shows some example dot-movement scenarios extracted from the processing of actual halftone images. The contours represent the level sets of the different cost functionals. Note how the form of the cost functional depends upon the locations of the neigboring dots, which are inside circles. We have viewed several example dot-movement scenarios. For the most part, the cost functionals are approximately convex, with minor non-convexities occuring around the locations of the neighboring dots, as shown in Figs. 3 (a) and (b). But generally, the cost functionals can have considerably more irregular shapes. Moreover, they can have multiple local minima, as shown in Fig. 3 (c). In any case, since the minimization is greedy, the dot being moved always ends up at the minimum of the "watershed" that contains its initial location. Of course, if the dot being moved has its initial location at a local minimum, then it is not moved at all, as shown in Fig. 3 (d).

3.2 Finding neighboring dots

It is critical to choose the N neighboring dots so that the resulting dot distribution is homogeneous. For this, we choose one neighboring dot from each of N separate sectors subtending equal angles, as illustrated in Fig. 4. This has the effect of forcing dots to mingle. Otherwise, isolated groups of at least $N + 1$ dots could permanently congregate in self-stable "cliques." (We actually observed this in early experiments, when we were using a different neighbor-selection procedure.) To further encourage isotropic dot distributions, we orient the boundaries of the sectors randomly for each dot. This too is illustrated in Fig. 4. We tried several other methods for choosing the neighboring dots; but the procedure described here gives the best results.

Note that since we find the neighboring dots in separate sectors, there may be other neighboring dots which are close to the dot being moved, but which are not included in the cost functional. Practically, the only potentially bad consequence of this would be that the dot could end up adjacent to another dot. This would actually be a serious problem, since an adjacent pair of minority pixels

in a region of extreme tone is surprisingly visible. To prevent this, we enforce the rule that the dot is not allowed to move into any location that has another dot as a neighbor.

Finally, an important parameter is the number N of neighboring dots which are chosen. We have used a value of $N = 4$ to obtain our most homogeneous, isotropic dot distributions.

We found that two iterations of the dot rearrangement is sufficient for good results. More iterations will increase the smoothness and homogeneity of the dot distribution. However, this may not be worth the extra computation.

4 Generating the edge map

Before the dot-rearrangement stage, we generate an edge map E that assigns the value 1 to pixel locations close to edges, and the value 0 to all other pixel locations. It is used to disable postprocessing around edges, which can be rendered diffuse and soft by the dot rearrangement process. Other approaches to finding edges in binary images have been proposed [18]. However, the approach that we describe here is specifically geared toward finding edges in regions of extreme tone. An overview of our edge map generation process is illustrated in Fig. 5.

We obtain E as the logical "or" of two separate edge maps: one identifying edges in light regions, and the other identifying edges in dark regions. Formally, for pixel location n,

$$E(n) = \begin{cases} 1 & \text{if } E_l(n) = 1 \text{ or } E_d(n) = 1 \\ 0 & \text{else} \end{cases} , \tag{5}$$

where E_l and E_d are edge maps for the light and dark regions, respectively.

The edge maps E_l and E_d are obtained from lower-resolution edge maps e_l and e_d using block replication. For this reason, the high-resolution edge map E often has a blocky appearance.

To generate e_l and e_d, we use a lower-resolution version of the input halftone called the decimated image array. By convention, we define each element in this array to be the number of black dots in an $(L \times L)$–pixel block of the halftone image, where L is the subsampling factor (see Fig. 6). This means that the decimated image array takes values in $\{0, \ldots, L^2 - 1\}$.

To construct e_l, we assign a 1 to any location in the decimated image array which is part of a 2×2 block where either a horizontal or vertical edge was detected, and a 0 to all other locations. To detect edges in a block, we apply edge operators to the block, each of which yields a scalar output as illustrated in Fig. 7. The magnitudes of the scalars are compared to a locally adaptive threshold τ; and if one of them is greater than the threshold, then we say that an edge is present. The threshold τ is computed as

$$\tau(\sigma) = K_1\sigma + K_2 , \tag{6}$$

where σ is the sum of the values in the 2×2 block, and K_1 and K_2 are nonnegative constants. For the block outlined in bold in Fig. 7, for example, the threshold τ would take the value of $K_1(3 + 4 + 0 + 0) + K_2$, or $K_1 \cdot 7 + K_2$.

To construct e_d, the procedure is essentially the same. The only difference is that instead of using the decimated image array, we use the inverted decimated image array, which is computed as

$$D^{\text{inv}} = L^2 - 1 - D , \tag{7}$$

where D and D^{inv} denote the decimated image array and the inverted decimated image array, respectively.

Note from (6) how the edge-detection threshold τ depends linearly on σ. In the case of generating e_l, for example, the result of this is that the edge detection process is most sensitive to edges in the

lightest regions; and it gets less sensitive as the tone decreases in lightness toward the midtones. (An analogous situation exists for the case of generating e_d.) The adaptive threshold always performs at least as well as a fixed threshold would, since that is simply the same as constraining K_1 to equal 0. In a majority of our experiments, the best result achievable by allowing both K_1 and K_2 to vary is about the same as the best result achievable by fixing K_1 to be 0. However, in some cases, our adaptive-thresholding approach can perform appreciably better.

We found that the best way to obtain values for K_1 and K_2 is to experiment with ranges of values for these variables, and then to pick the pair that yields the best result. Certainly, it would have been preferable to find a fixed pair (K_1, K_2) that is suitable for all images. But short of this, we do recommend choosing values for K_1 from $[0.0, 0.6]$, and for K_2 from $[2.0, 8.0]$.

An important parameter in the edge detection process is the block size L for the decimated image array. If L is too large, then the resulting edge map is too coarse. However, if L is small, then the dynamic range of the decimated image array will also be small. This can degrade the edge detection by limiting the range of values that the edge operators can possibly yield. After experimenting with a wide range of images, we found that a value of $L = 8$ yields an edge map that identifies edges adequately without being too coarse.

5 Results

We tested the *Springs* algorithm by applying it to images acquired from various sources and halftoned using both error diffusion and screening. In this section, we first show the results of applying *Springs* to three different images. Then we demonstrate some shortcomings of the *Springs* algorithm. All images in this paper are reproduced at 100 dots per inch (dpi).

Figures 8, 9, and 10 show results generated using three different images. The image for Fig. 8 was scanned from a magazine and halftoned by error diffusion; the image for Fig. 9 was taken from a photo CD library and halftoned by error diffusion; and the image for Fig. 10 was created in Photoshop and screened using a 128×128 dispersed-dot dither array. Here, by "error diffusion," we refer to the instantiation introduced in [2]; and to design the dither array for screening, we used the algorithm described in [6]. The *Springs* algorithm could equally well have been applied to halftones generated by other versions of error diffusion (see, for example, [19], [5], [20], or [4]), or by screening using dither arrays generated from other methods (see, for example, [21], [22], [23], or [8]).

In each figure, image (a) shows the input halftone, and image (b) shows the output of the *Springs* algorithm obtained using the parameters shown in Table 1. Note that in the light regions and in the dark regions, the minority pixels have been redistributed for a smoother, more homogeneous appearance. Image (c) in each figure shows the edge map, with black representing the regions where dot rearrangement was not allowed due to the detection of edge structures. To see why this is important, refer to image (d), which shows the result of rearranging dots while ignoring the edge segmentation. It is clear that if the edge segmentation is not used, then major edges in highlight and shadow regions — which are of critical importance when it comes to image quality — can be rendered soft and diffuse.

Since the basic action of *Springs* is to rearrange selected dots, two types of problems can occur: dots can be moved when they should not have been moved; or they can remain fixed when they should have been moved. These two scenarios are demonstrated in Fig. 11. To generate this figure, we first created a continuous-tone input image in Photoshop with the deliberate intent of "breaking" the *Springs* algorithm. The results of applying the *Springs* algorithm for this image appear for error diffusion on the left, and for dispersed-dot screening on the right. (Again, for error

diffusion, we used [2]; and for the dither array generation, we used [6].) For the edge segmentations, we used a value of $L = 8$ for the block size; $K_1 = 0.0$, $K_2 = 6$ for the error diffusion result, and $K_1 = 0.2$, $K_2 = 2$ for the screening result. (These were the best sets of parameters that we could find for these images.) Note that for the halftone generated by error diffusion, some "worming" artifacts in the background were detected as edges and actually protected from the postprocessing. In both results, the edge between the light background and the darker region in the upper, left-hand corner was not detected completely. Wherever that edge was not detected, the dots of the darker region appear to have "leaked" into the background as a result of the pixel rearrangement.

6 Conclusions

In this paper, we introduced *Springs*, an algorithm for improving halftone appearance by rearranging minority pixels in highlight and shadow regions. *Springs* moves a dot by first defining a cost functional based on the locations of neighboring dots, and then moving the dot to a local minimizer of the cost functional. This has the effect of relaxing the dot distribution. Since the dot-movement procedure has the effect of softening edges, it is only allowed in regions where edges are not detected. We demonstrated the effectiveness of the *Springs* algorithm on halftone images generated using both error diffusion and screening.

References

[1] R. A. Ulichney, *Digital Halftoning*. Cambridge, MA: MIT Press, 1987.

[2] R. Floyd and L. Steinberg, "An adaptive algorithm for spatial greyscale," *Proc. SID*, vol. 17, no. 2, pp. 75 – 77, 1976.

[3] B. E. Bayer, "An optimum method for two-level rendition of continuous-tone pictures," *Proc. of IEEE Int'l Conf. on Comm.*, 1973, pp. 26.11 – 26.15.

[4] R. A. Ulichney, "Dithering with blue noise," *Proc. IEEE*, vol. 76, pp. 56 – 79, 1988.

[5] R. Levien, "Output dependent feedback in error diffusion halftoning," *Proc. of IS&T's* 46^{th} *Annual Conference*, May 1993, Cambridge, MA, pp. 115 – 118.

[6] R. A. Ulichney, "The void-and-cluster method for dither array generation," *Proc. of SPIE/IS&T Symp. on Electronic Imaging Science and Tech.*, February 1993, San Jose, CA, pp. 332 – 343.

[7] Q. Lin, "Improving halftone uniformity and tonal response," *Proc. of IS&T's Tenth Int'l. Congress on Advances in Non-Impact Printing Technologies*, November 1994, New Orleans, LA, pp. 377 – 380.

[8] J. P. Allebach and Q. Lin, "FM screen design using DBS algorithm," *Proc. of the 1996 IEEE Int'l Conf. on Image Proc.*, vol. 1, 1996, pp. 549 – 552.

[9] R. Eschbach, "Pulse-density modulation on rastered media: combining pulse-density modulation and error diffusion," *J. Opt. Soc. Am. A*, vol. 7, pp. 708 – 716, 1990.

[10] P. W. Wong, "Entropy-constrained halftoning using multipath tree coding," *IEEE Trans. on Image Processing*, vol. 6, no. 11, pp. 1567 – 1579, 1997.

[11] R. Eschbach and R. Hauck, "A 2-D pulse density modulation by iteration for halftoning," *Opt. Comm.*, vol. 62, pp. 300 – 304, 1987.

[12] M. Analoui and J. P. Allebach, "Model-based halftoning using direct binary search," *Proc. of SPIE/IS&T Symp. on Electronic Imaging Science and Tech.*, February 1992, San Jose, CA, pp. 109 – 121.

[13] J. B. Mulligan and J. A. J. Ahumada, "Principled halftoning based on models of human vision," *Proc. of SPIE/IS&T Symp. on Electronic Imaging Science and Tech.*, February 1992, San Jose, CA, pp. 96 – 108.

[14] T. N. Pappas and D. L. Neuhoff, "Least-squares model-based halftoning," *Proc. of SPIE/IS&T Symp. on Electronic Imaging Science and Tech.*, February 1992, San Jose, CA, pp. 165 – 176.

[15] C. B. Atkins, J. P. Allebach, and C. A. Bouman, "Halftone postprocessing for improved highlight rendition," *Proc. of the 1997 IEEE Int'l Conf. on Image Proc.*, vol. 1, 1997, pp. 791 – 794.

[16] J. C. Handley and E. R. Dougherty, "Model-based optimal restoration of fax images in the context of mathematical morphology," *J. Electronic Imaging*, vol. 3, no. 2, pp. 182 – 189, 1994.

[17] E. R. Dougherty and R. P. Loce, "Optimal binary differencing filters: design, logic complexity, precision analysis, and application to digital document processing," *J. Electronic Imaging*, vol. 5, no. 1, pp. 66 – 86, 1996.

[18] K.-C. Fan and C.-C. Han, "Parallel mechanism for detecting contours in binary images," *J. Electronic Imaging*, vol. 3, no. 1, pp. 30 – 36, 1994.

[19] B. Kolpatzik and C. A. Bouman, "Optimized error diffusion for image display," *J. of Electronic Imaging*, vol. 1, no. 3, pp. 277 – 292, 1992.

[20] P. W. Wong, "Adaptive error diffusion and its application in multiresolution rendering," *IEEE Trans. on Image Processing*, vol. 5, pp. 1184 – 1196, 1996.

[21] T. Mitsa and K. J. Parker, "Digital halftoning technique using a blue noise mask," *J. Opt. Soc. Am. A*, vol. 9, pp. 1920 – 1929, 1992.

[22] J. R. Sullivan and L. A. Ray, "Digital halftoning with correlated minimum visual modulation patterns," 1993. United States Patent 5,214,517.

[23] R. A. Ulichney, "Filter design for void-and-cluster dither arrays," *SID 94 digest of technical papers*, June 1994, San Jose, CA, pp. 809 – 812.

Figure	L	K_1	K_2
8	8	0.0	8
9	8	0.2	6
10	8	0.0	8

Table 1: Edge segmentation parameters used for Figs. 8–10.

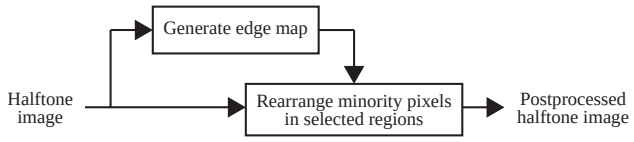


Figure 1: Structure of the *Springs* algorithm.

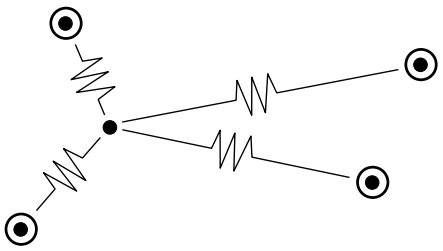


Figure 2: Central dot connected to neighboring dots by springs. (Neighboring dots are inside circles.)

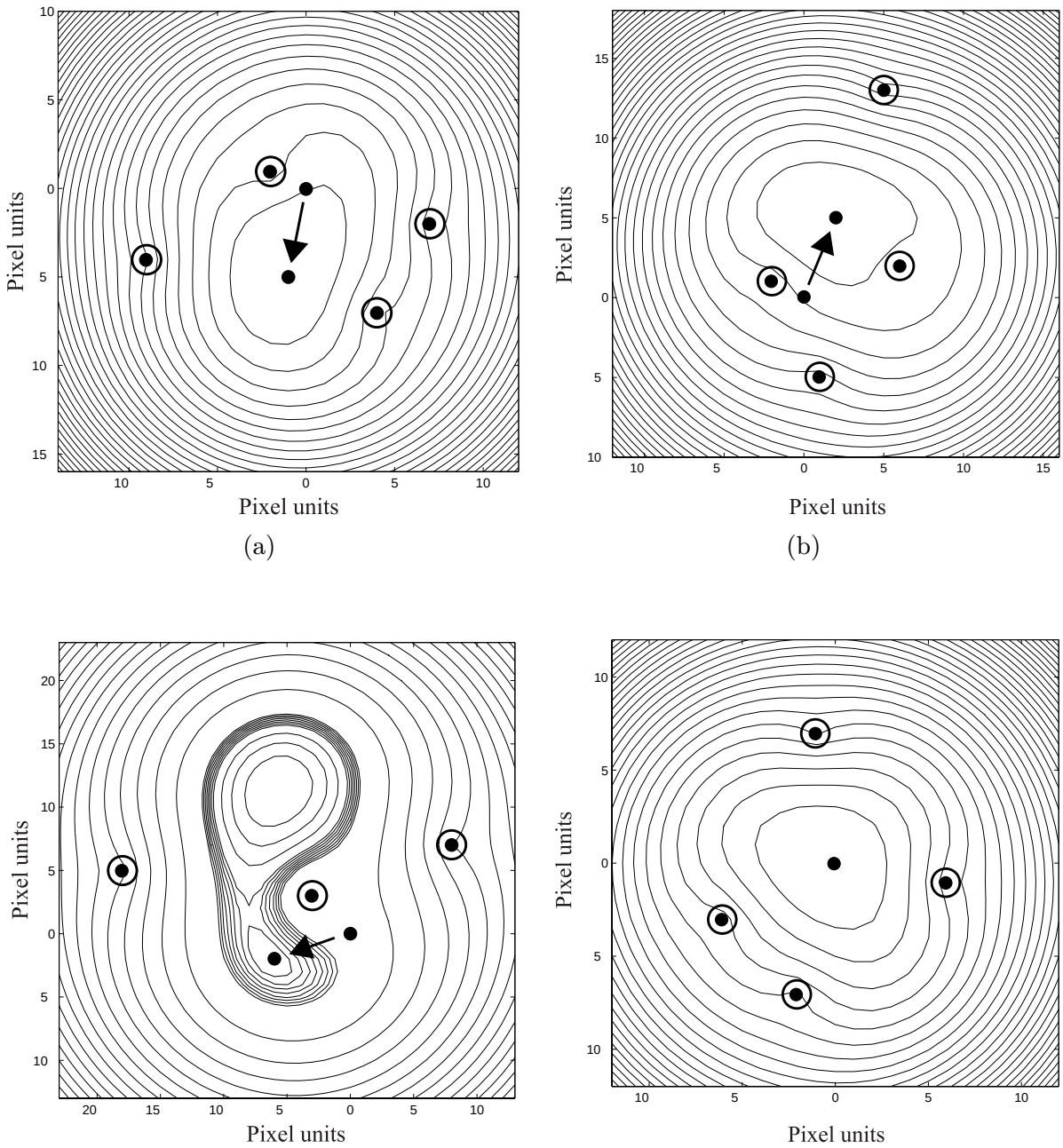


Figure 3: Example dot-movement scenarios. Curves represent level sets of the cost functional, which is defined by the relative positions of neighboring dots. (Neighboring dots are inside circles.)

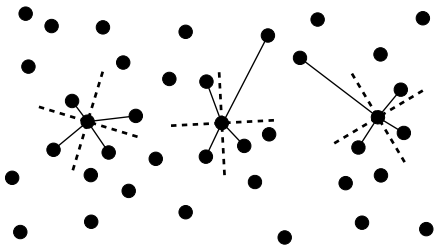


Figure 4: One neighboring dot is found in each sector. Boundaries of sectors are oriented randomly.

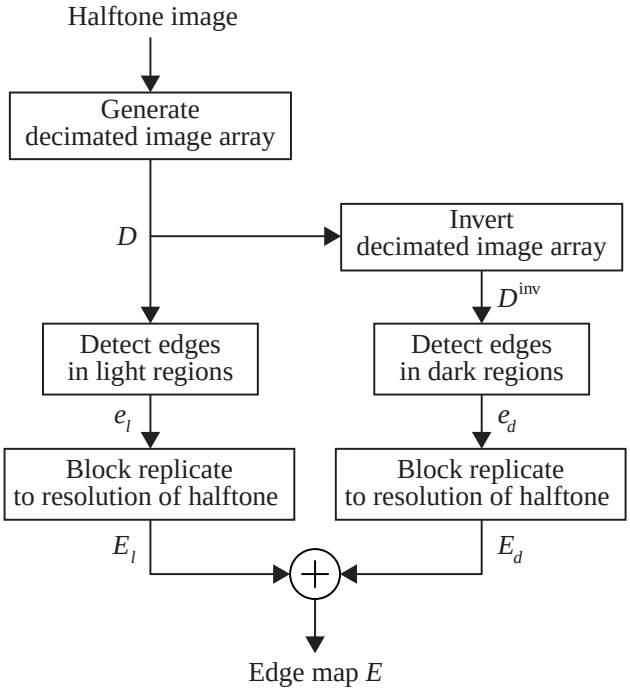


Figure 5: Edge map generation process.

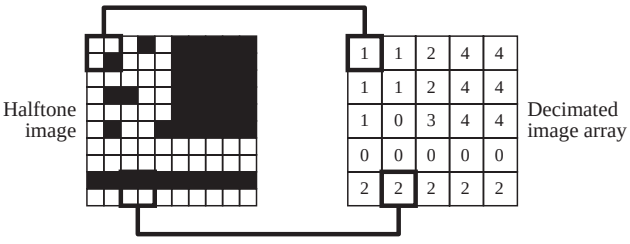


Figure 6: Construction of the decimated image array with a subsampling factor of $L = 2$.

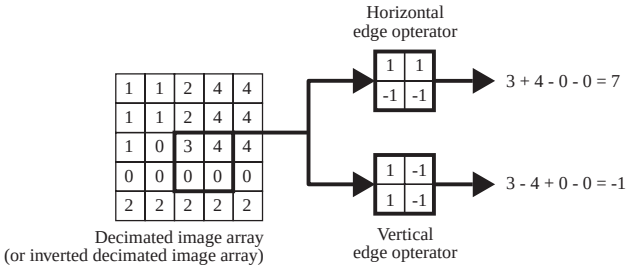
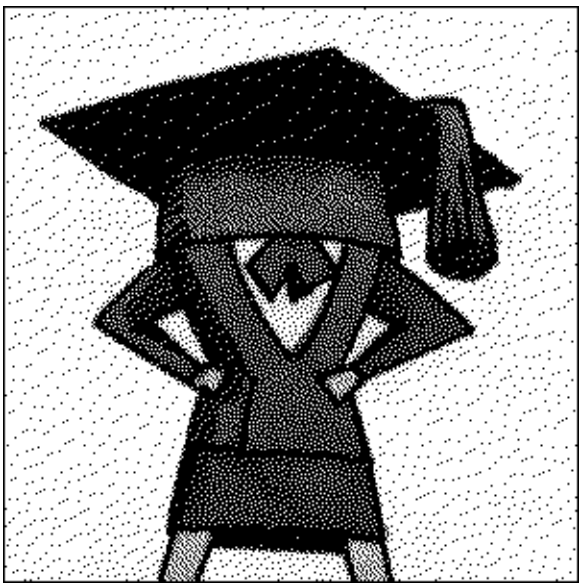
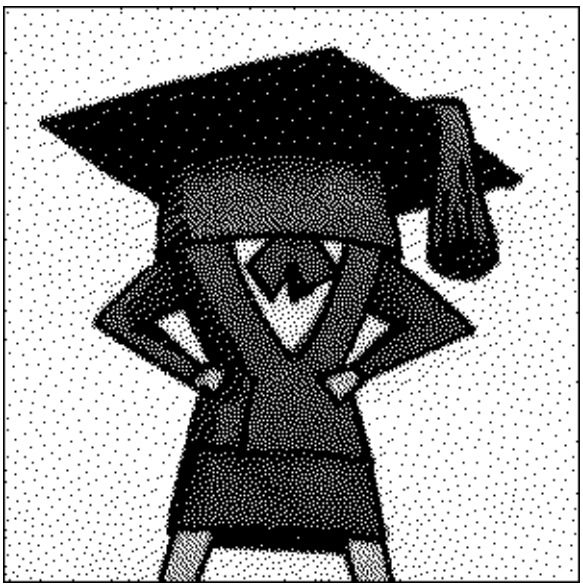


Figure 7: Edge-detection operators.

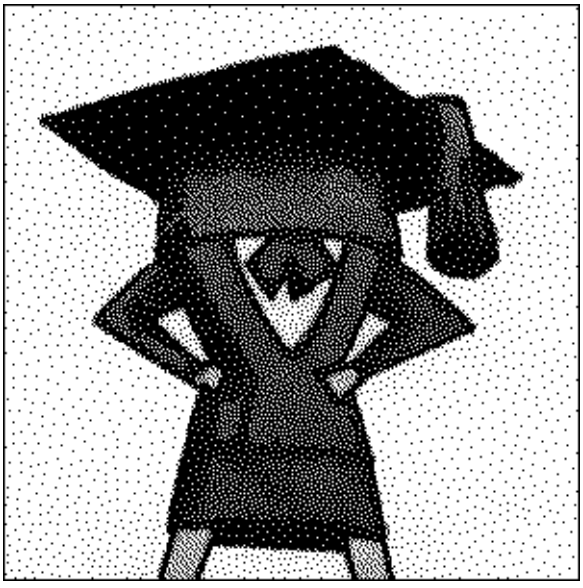


(a) Input image scanned from a magazine advertisement, then halftoned using error diffusion.



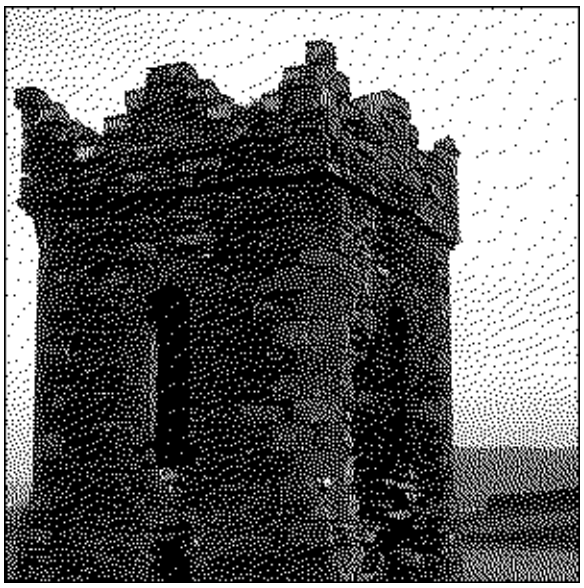
(b) Image (a) after postprocessing by the *Springs* algorithm.

(c) Segmentation generated by *Springs* while processing image (a).

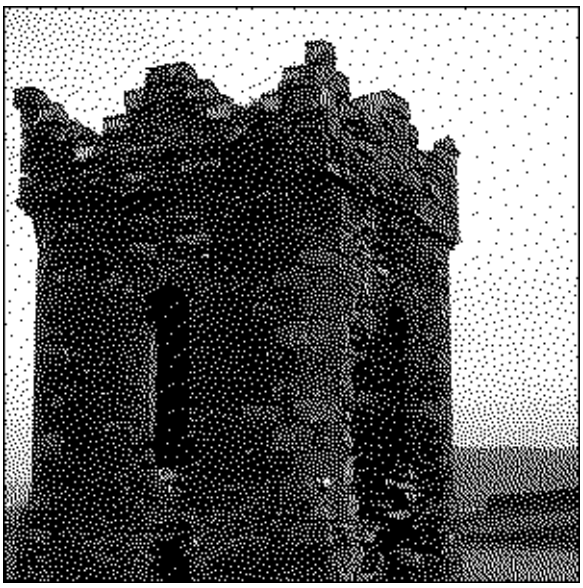


(d) Image (a) after postprocessing, ignoring the segmentation.

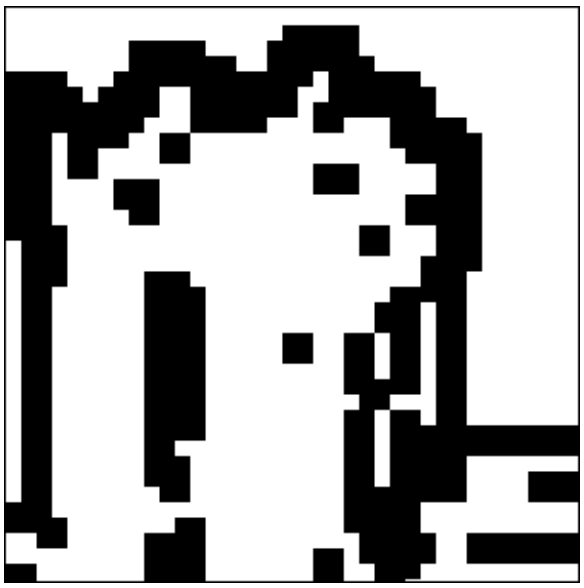
Figure 8: Postprocessing results for a scanned image halftoned using error diffusion.



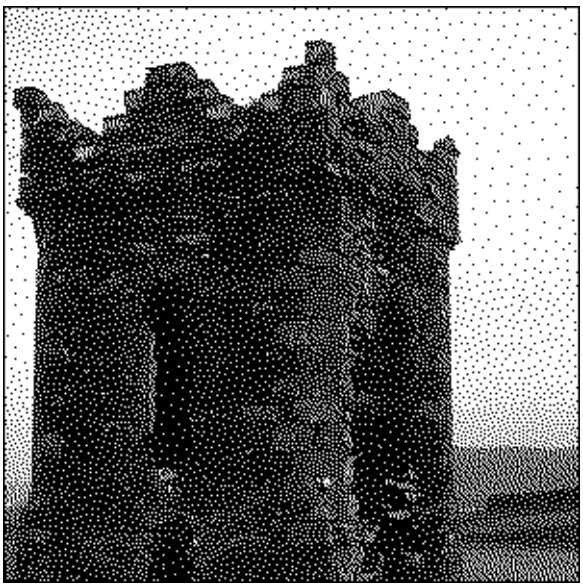
(a) Input image acquired from a photo CD library, then halftoned using error diffusion.



(b) Image (a) after postprocessing by the *Springs* algorithm.

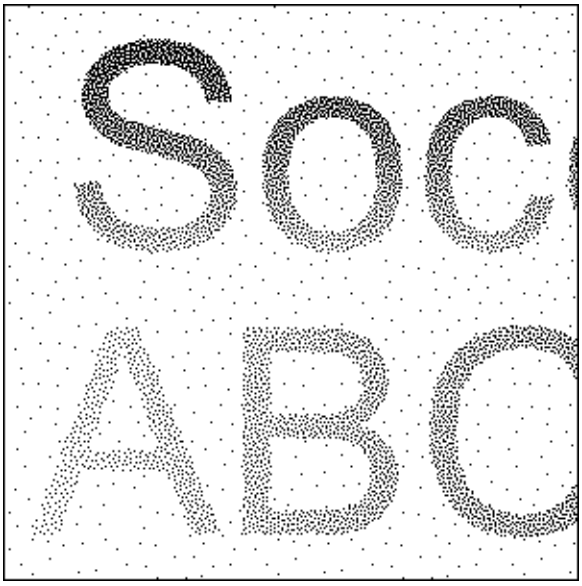


(c) Segmentation generated by *Springs* while processing image (a).

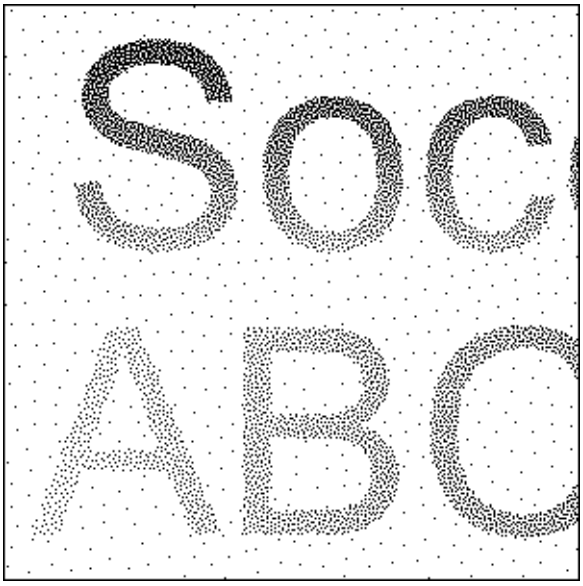


(d) Image (a) after postprocessing, ignoring the segmentation.

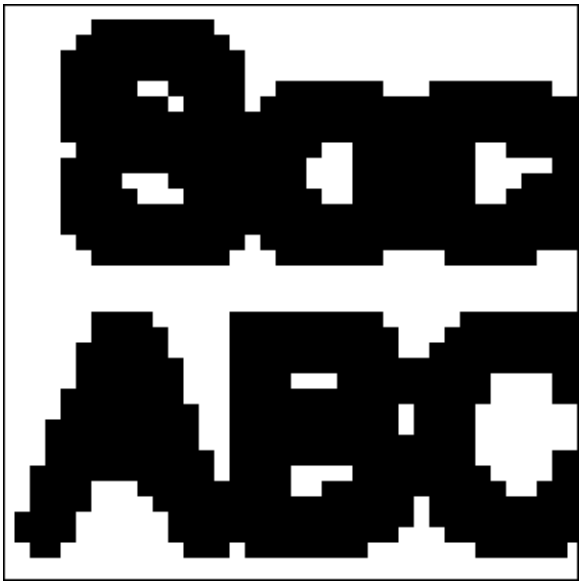
Figure 9: Postprocessing results for an image halftoned using error diffusion.



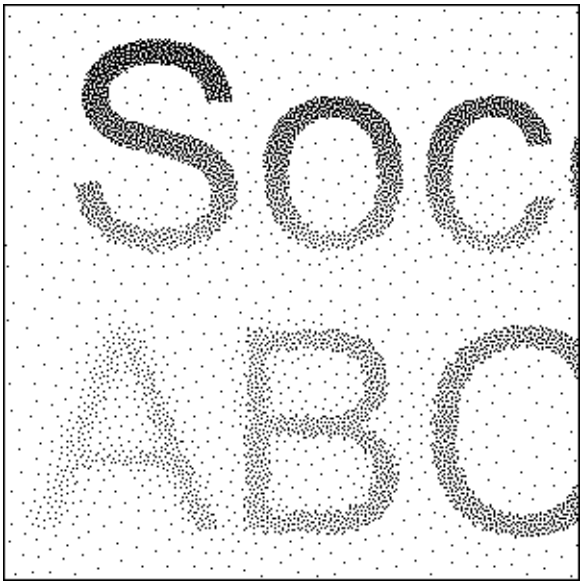
(a) Input image generated using Photoshop, then halftoned by dispersed-dot screening.



(b) Image (a) after postprocessing by the *Springs* algorithm.



(c) Segmentation generated by *Springs* while processing image (a).



(d) Image (a) after postprocessing, ignoring the segmentation.

Figure 10: Postprocessing results for a synthetic image halftoned by dispersed-dot screening.

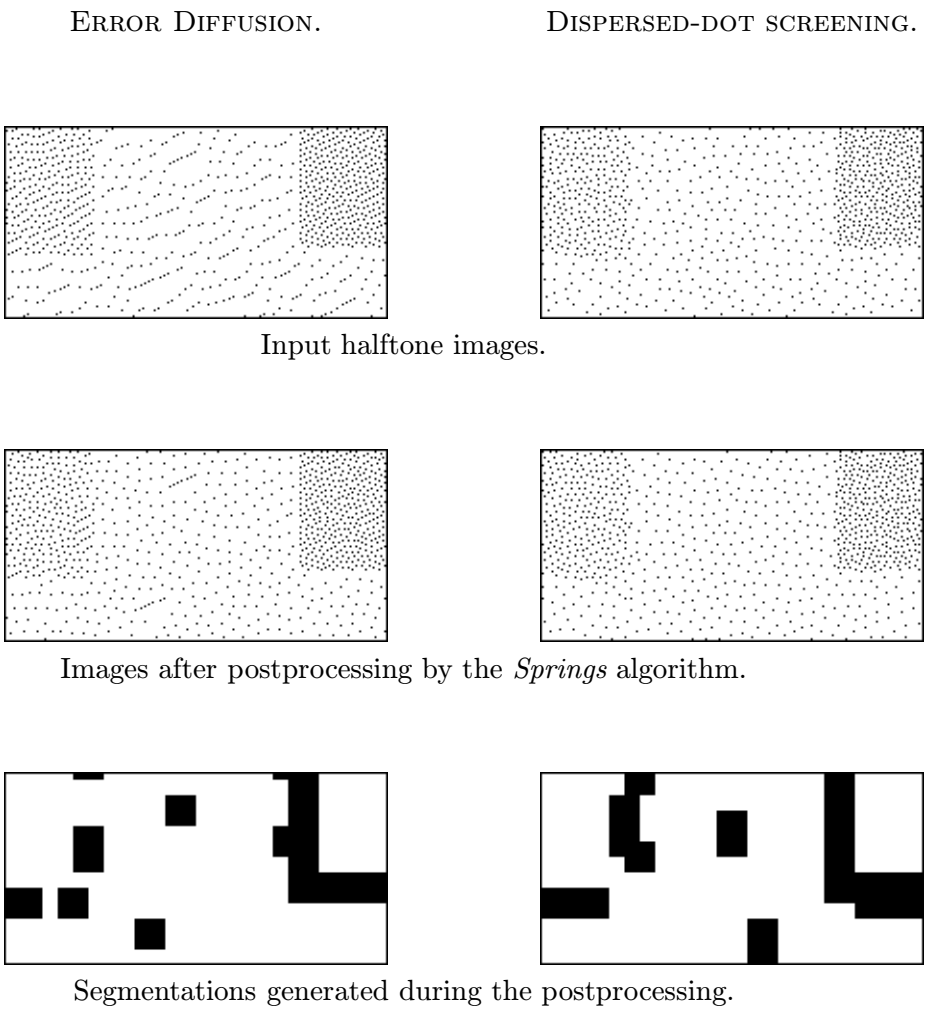


Figure 11: Demonstration of some shortcomings of the *Springs* algorithm. The input images were generated by first using Photoshop to create the continuous-tone original, then halftoning by error diffusion (left column) and dispersed-dot screening (right column). (This image was generated with the specific objective of breaking the *Springs* algorithm.)