Adaptive Thresholding for the DigitalDesk

Pierre D. Wellner
wellner@europarc.xerox.com

Introduction to the problem

The image produced by a video camera pointing at black printing on a white sheet of paper is not truly black and white. This image is gray-scale (or color) no matter where the camera is pointed. Unless lighting on the desk is carefully controlled, video images of paper lying on the desk are poor representations of the originals. Unlike the inside of a scanner or photocopier, it is very difficult to guarantee even lighting across the surface of a desk. This open space may be exposed to desk lamps, overhead lights, windows or moving shadows, all of which can cause varying brightness across the page. Human vision compensates for this, but when the image of paper on the desk is manipulated by machine without taking these variations into account the results can be very bad. In Figure 1, for example, the background of the right hand “in case of emergency” box does not match the surrounding background, leaving a distinct edge. The box on the right appears

Figure 1: copy & paste with an unevenly lit gray-scale image.

lighter than the box on the left, even though it is an exact copy (fold this page if you want to check). The only difference is in the brightness of the surrounding background. It gets darker from left to right because the light was on the left side of the paper from which this image was grabbed,

This problem is most noticeable when dealing with high contrast line art or text because the original document is genuinely black and white, yet the camera produces an image of varying levels of gray. Many applications prototyped on the DigitalDesk must clearly determine which parts of the image are meant to be black or white in order to project line art back onto the desk, or to pass text images to an optical character recognition (OCR) server. The system cannot use the gray images (typically 8 bits per pixel), so it must convert them to black and white images (one bit per pixel). There are many ways to do this. In some cases, when the resulting black and white image is meant to be viewed by people, it should look as much as possible like a gray-scale image and the image is *dithered* or *half-toned* (using a technique such as [3]) to produce a result like Figure 1. Although this image looks like a gray-scale image, it is made up of small black and white spots of varying patterns, densities and sizes. Human vision makes sense of these images almost as well as the gray images and the original black and white images. But for machine processing operations, such as character recognition, select & paste operations, and merging together of many different images, the system cannot use dithered or half-toned images. Although only one bit per pixel, these images are even less suitable for machine processing than the original gray-scale images. The system needs simple patterns of lines, characters, and relatively large patches of black and white. The process by which these types of black and white images are generated from gray images is commonly referred to as *thresholding*, and it is an important requirement for any digital desk system.

There are many ways to threshold an image, but the basic process is to examine each gray pixel and decide whether to make it either black or white. This report describes the various techniques that were developed and tested for thresholding on the DigitalDesk, and it ends with the description of an algorithm that was found to be suitable. This algorithm (here called "quick adaptive thresholding") may not be the best possible, but it works well enough that other problems of the DigitalDesk became more important when the work reached the stage described here.

Global Thresholding

In a way, thresholding can be seen as an extreme form of *contrast enhancement*, or making light pixels lighter and dark pixels darker. The simplest (and most common) way to threshold an image is to set all pixels below a certain gray-level to black, and to clear all others to white. The question then is how to select this gray-level. One possibility is to pick the center of the range of possible values, so if pixels are eight bits deep (ranging from 0 to 255), for example, then 128 would be selected. This approach works well if all the "dark" pixels of the image really do have values under 128, and light pixels have values above 128, but if the image is over or under exposed, then the result might be all white or black. It is better to look at the range of *actual* values instead of *possible* values to determine the threshold. The maximum and minimum values for each pixel in the image can be found, then the midpoint used as the threshold. An even better way to select the threshold is not just to look at the range of actual values, but also their distribution. If, for example, you expect the image to be of black line art, or text on a white background then you expect that most pixels will be the intensity of the background and a smaller but significant proportion will be of the dark ink. A histogram of the pixel intensities should look something like Figure 2.

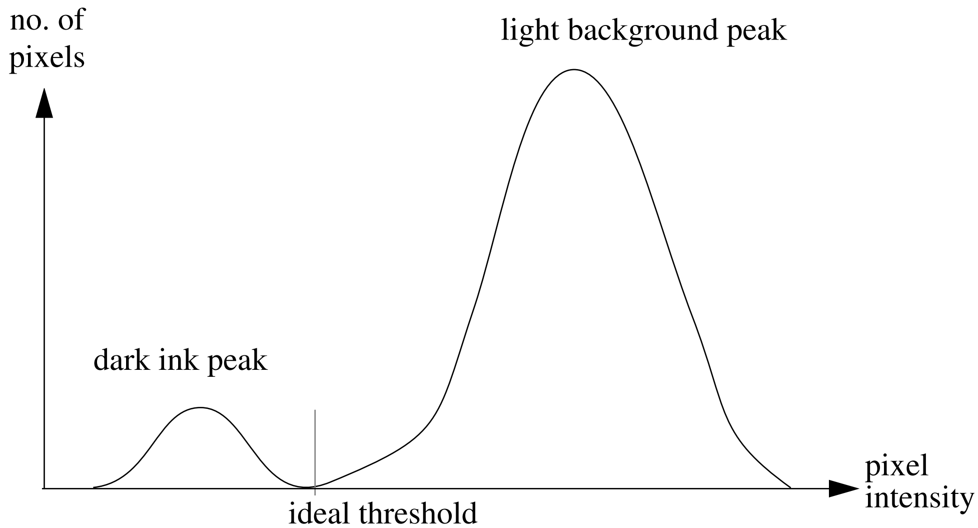
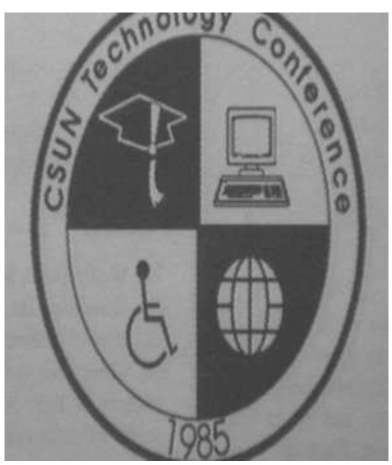


Figure 2: theoretical histogram

A large background peak should be visible, as well as a smaller peak for the dark ink. The whole curve might be shifted to the left or right depending on the ambient light level, but in any case, the best value to pick for thresholding is the local minimum between the two peaks. This is fine in theory, but how well does it work in practice? Figure 3 shows a frame-grabbed image and its histogram for which the technique works well. The smoothed histogram shows two prominent peaks,



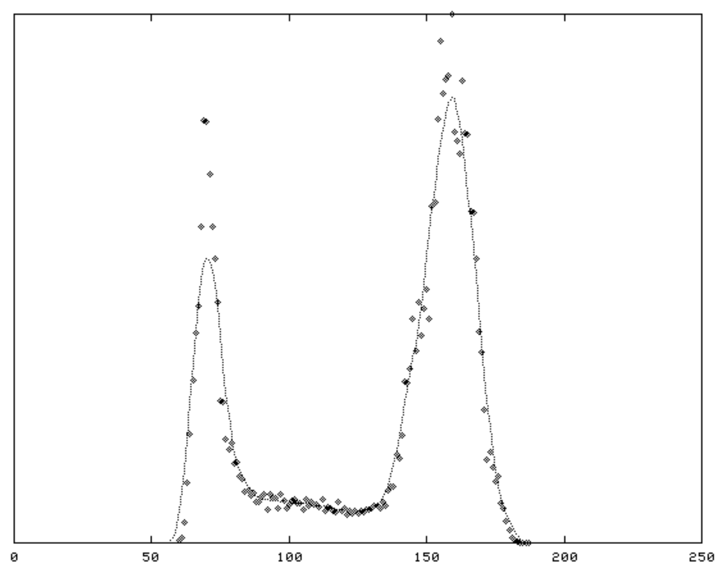


Figure 3: Grabbed image with clearly bimodal histogram

and it would not be difficult to calculate a near-ideal threshold by fitting a curve to this histogram or even taking the midpoint between the two peaks. This is not a typical image, however, because it has large amounts of both black and white. The DigitalDesk must also be able to threshold images like the one in Figure 4. In the histogram of this image, the smaller “dark” peak is almost lost in the noise, so it is impossible to reliably locate a local minimum between peaks.

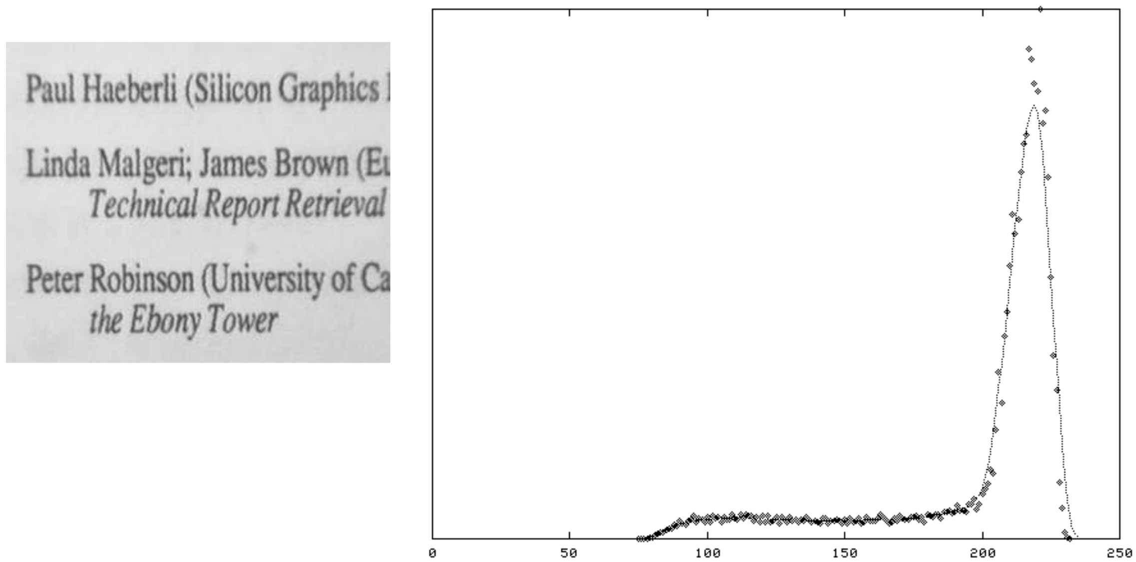
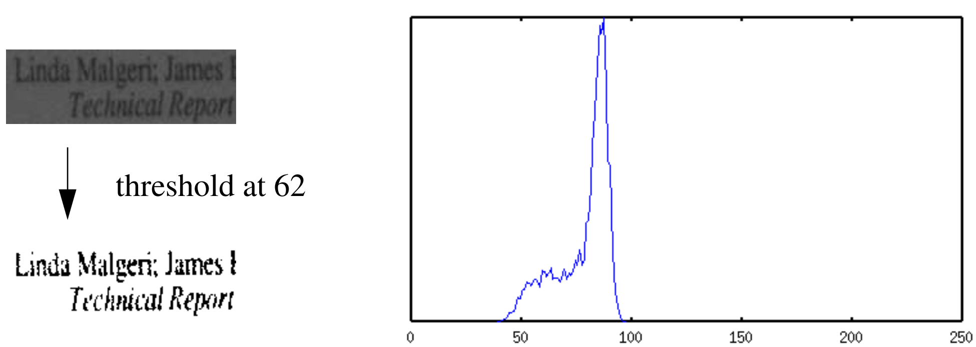


Figure 4: Grabbed image with "dark" peak almost lost in the noise

In any case, a large (background) peak is always present and easy to find, so a useful strategy for thresholding an image is the following:

1) Calculate the histogram.
2) Locate the large peak by finding the maximum value of a moving average of the histogram curve. The moving average smooths out the effects of noise on the maximum value, as shown in Figure 3 and Figure 4.
3) Select the threshold value at a certain proportion of the distance between this peak and the minimum value.

Experimentation seems to indicate that 1/2 of this distance produces quite good results in a broad range of lighting conditions ranging from very bright to almost completely dark. The figures below show an image grabbed under four different lighting conditions and the result from applying this histogram-based global thresholding algorithm to each. Despite the wide range of lighting (as can be seen from the histograms) this algorithm selects an appropriate threshold in each case, and the thresholded result for each is almost identical.



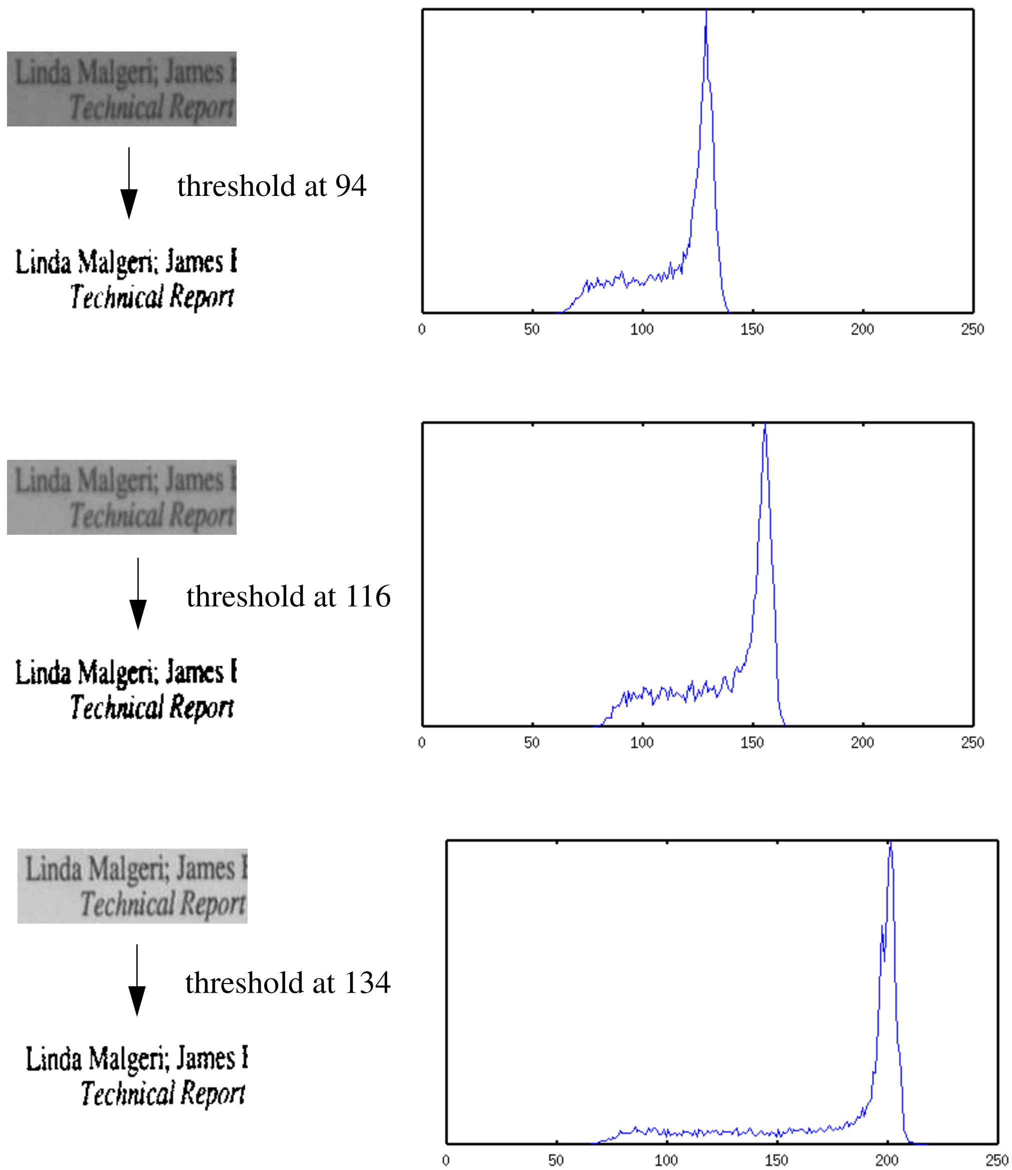


Figure 5: histogram-based global thresholding at various intensities

This histogram-based global thresholding technique works quite well for images that are evenly lit or, as in the examples above, for images that represent very small parts of paper where the light does not vary much. But it fails to produce good results for larger areas and in normal office lighting where the range of brightness varies greatly across the desk. Because a single global threshold value is used for the entire image, some parts of the image become too white and others too dark.

Much of the text then becomes unreadable, as illustrated in Figure 6.

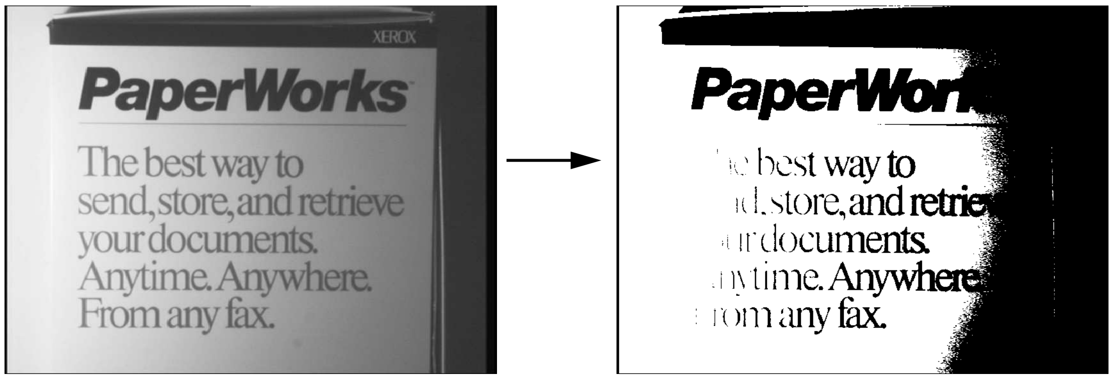


Figure 6: global thresholding of unevenly lit image

Producing good thresholded images from paper that is unevenly lit requires an *adaptive thresholding* algorithm. This technique varies the threshold value across the image according to the background illumination at each pixel. The discussion below is illustrated with the results of algorithms applied to the image in Figure 6. This is a challenging test because the image is lit from the side, and it has black text on a white background (the word “PaperWorks”), white text on a black background (“XEROX”), gray text on a white background (“The best way...”), various shadows, and a thin horizontal black line under the word “PaperWorks.”

Adaptive Thresholding

An ideal adaptive thresholding algorithm would produce the same result when applied to an unevenly lit page as a global thresholding algorithm would produce when applied to a perfectly evenly lit page. The brightness of each pixel is normalized to compensate for being more or less illuminated, and only then is a decision made as to whether the result should be black or white. The question then, is how to determine the background illumination at each point. One simple way to do this is to grab the image of a blank page before grabbing the page to be thresholded. This blank page can then be used as a reference image. For each pixel to be thresholded, the value of the corresponding pixel in the reference image is subtracted before applying the threshold.

This method produces very good results as long as the lighting does not change between the time when the reference image and subject image are each grabbed. On a desk, however, the lighting changes frequently as the user's shadow moves across the desk or doors, lamps and other objects are moved about. If there is a window in the room, then the lighting also changes throughout the day. One solution might be to immediately precede each grab with a reference grab of a blank page at the same location, but this would be almost as inconvenient to use as a scanner, thus defeating an important reason for using an overhead video camera in the first place.

Another solution is to try to estimate the background illumination at each pixel by making some assumptions about what the image should look like. We could assume, for example, that the image is mostly background (e.g. white), and that dark markings make up a smaller part of the image. Another assumption we could make is that the background lighting changes relatively slowly across the page compared to the changes in light and dark due to marks on page. Many algorithms based on such assumptions are possible. There is no mathematical theory of adaptive thresholding, and as a consequence, there is also no standard or optimal method for doing it [5] (p. 597). Instead, there are several ad hoc methods, some of which are more popular than others. Because the methods are ad hoc, it would be useful to be able to measure their performance. For this, Haralick and Shapiro [4] propose the following guidelines: regions should be uniform and homogeneous with respect to gray tone; region interiors should be simple and without many small holes; adjacent regions should have significantly different values; and boundaries of each segment should be simple, not ragged, and must be spatially accurate.

According to Pratt, no quantitative performance metric has been developed for image thresholding. It seems the main way to evaluate an algorithm is simply to look at the result and judge whether it looks right. For images of text, there is a quantitative possibility. The result of various algorithms under various lighting conditions could be fed to an OCR system, and the text produced from this can be compared to the original text. Although potentially useful, this approach was not systematically used for the algorithms described below, because it was deemed unlikely to give a different evaluation from the "looks better" judgement.

For interactive applications of the DigitalDesk, users must wait for thresholding to finish when completing interaction techniques such as "copy and paste" of text or graphics. So another important quantitative performance metric is speed. The following sections describe various adaptive thresholding algorithms and show the results they produce.

Adaptive thresholding based on Wall's algorithm

One technique for calculating a threshold that varies across the image according to background illumination was developed by R. J. Wall and is described in [1]. The following algorithm is loosely based on this description. First, it breaks up the image into smaller tiles and calculates a histogram for each tile. Based on the peaks of these histograms, a threshold is chosen for each tile. Then, every point in the image is assigned a threshold by interpolating between the values chosen for each tile. Figure 7 was generated from the same gray image as Figure 6, but using this technique. The image was divided into nine tiles (3x3) and for each tile a threshold was selected 20%

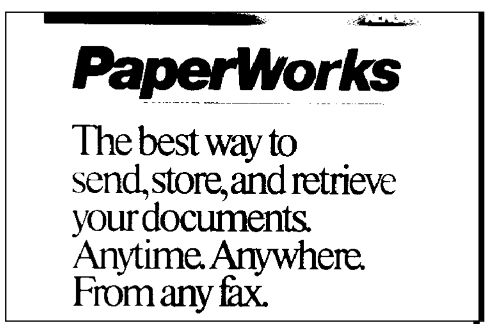


Figure 7: result of adaptive thresholding based on Wall's algorithm

below the peak. From these values, 16 tiles were formed with their corners at the center of each of the 9 tiles, and the threshold values were interpolated across each of these tiles from the corners. The result is much better than global thresholding, but because it requires more than one pass through the image, it is quite slow. Another problem with this technique is that with some images, the local histograms can be fooled by a large amount of black or white, causing the threshold not to vary smoothly across the image, and the result can be very bad (see in Figure 8.)

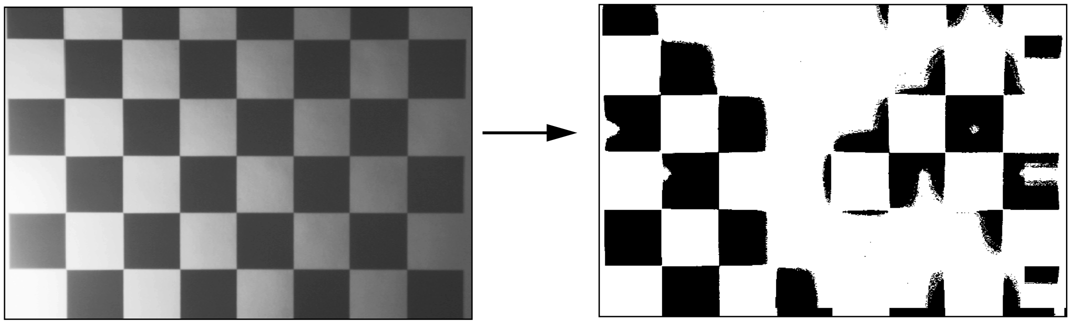


Figure 8: local histograms can be fooled

Quick Adaptive Thresholding

Most algorithms in the literature are more complex than Wall's algorithm, require more passes through the image and would take even longer to run *e.g*. [4, 5, 7]. It seems possible to implement a much simpler and faster algorithm for adaptive thresholding, so this section describes the steps taken so far in that direction for the DigitalDesk.

The basic idea is to run through the image while calculating a moving average of the last *s* pixels seen.When the value of a pixel is significantly lower than this average it is set to black, otherwise it is left white. Only one pass through the image is necessary, and the algorithm is simple enough to be implemented in hardware. It is interesting to note the similarities between the following algorithms and the one developed at IBM using analog hardware in 1968 [2].

Let p_n represent the value of a pixel at point *n* in the image being thresholded. For these purposes

					p_{n-s}	. . .	p_{n-3}	p_{n-2}	p_{n-1}	p_n				

we treat the image as though it were a single row of pixels composed of all the rows in the image lined up next to each other.

Let $f_s(n)$ be the sum of the values of the last *s* pixels at point *n*.

$$f_s(n) = \sum_{i=0}^{s-1} p_{n-i}$$

The value of the resulting image T*(n)* is either 1 (for black) or 0 (for white) depending on whether it is *t* percent darker than the average value of the previous *s* pixels.

$$\mathrm{T}(n) = \begin{cases} 1 \quad \text{if } p_n < \left(\frac{f_s(n)}{s}\right)\left(\frac{100-t}{100}\right) \\ 0 \quad \textit{otherwise} \end{cases}$$

Using 1/8th of the width of the image for the value of *s* and 15 for the value of *t* seems to yield the best results for a variety of images. Figure 9a shows the result of using this algorithm scanning the

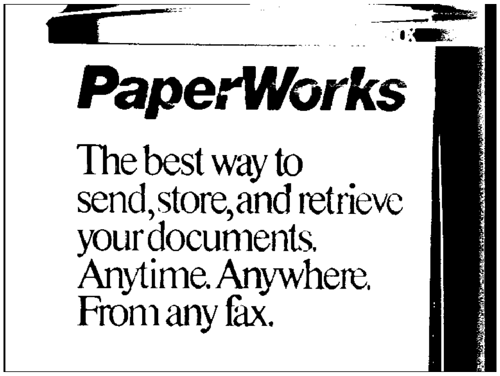


Figure 9a: moving average scanning from left to right.

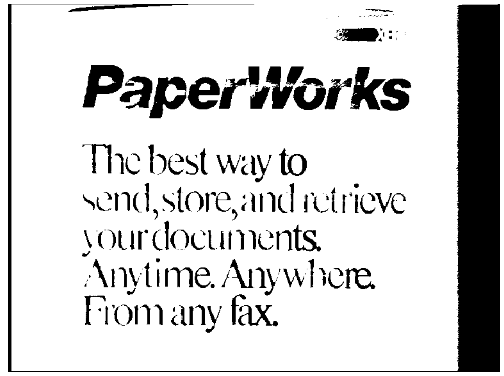


Figure 9b: moving average scanning from right to left.

rows of pixels from left to right. Figure 9b is exactly the same algorithm except the moving average scans from right to left. Notice in this image, that the left-most letters of the smaller text are incomplete, and there are more holes in the word PaperWorks. Also, the right-most black edge is much narrower. This is all because the background lighting of the image gets darker from left to right.

A fast way to calculate an approximate (weighted) moving average is to subtract $1/s$ part of it and add the value of only the latest pixel instead of using all s pixels. Thus $g(n)$ is an approximation of $f(n)$ where

$$\begin{aligned} g_s(n) &= g_s(n-1) - \frac{g_s(n-1)}{s} + p_n \\ &= p_n + \left(1 - \frac{1}{s}\right) \cdot g_s(n-1) \\ &= p_n + \left(1 - \frac{1}{s}\right) p_{n-1} + \left(1 - \frac{1}{s}\right)^2 p_{n-2} \cdots \\ &= \sum_{i=0}^{n} \left(1 - \frac{1}{s}\right)^i p_{n-i} \end{aligned}$$

The main difference between f(n) and $g(n)$ is that $g(n)$ puts more weight on the pixels that are closest to n, which is good (in fact the weighting function is exponential). Notice that if all values of p_n are the same, then $g(n) = f(n)$. For example, if $s = 2$, then

$$g_2(n) = p_n + \frac{p_{n-1}}{2} + \frac{g_2(n-2)}{4}$$

and $g_2(n) = f_2(n) = 2p_n$ because $g_2(n-2) = 2p_n$.

A remaining question is how to start the algorithm, or what value to use for $g(0)$. One possibility is to use sp_0, but because of edge effects, p_0 is not always a representative value, so another possibility is $127s$ (based on the midvalue for 8 bit pixels). In either case, this choice only affects the first few values of g. The weight of g(0) relative to the total of all weights used in calculating $g_s(n)$ is

$$\frac{\left(1-\frac{1}{s}\right)^{n}}{\sum_{i=0}^{n}\left(1-\frac{1}{s}\right)^{n}}$$

so if $s = 10$, for example, then for any $n > 6$, $g(0)$ contributes less than 10% of $g_{10}(n)$; for any $n > 22$, $g(0)$ contributes less than 1%. For s = 100, less than 10% happens after 8 pixels and less than 1% happens after 68 pixels.

The results from using this approximate moving average are similar (actually better) than the precise moving average, as can be seen in Figure 10a and Figure 10b.

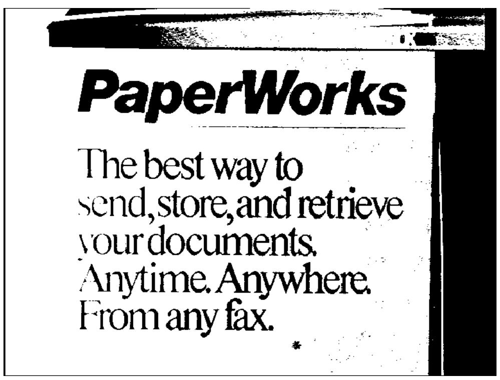


Figure 10a: approximate moving average scanning from left to right.

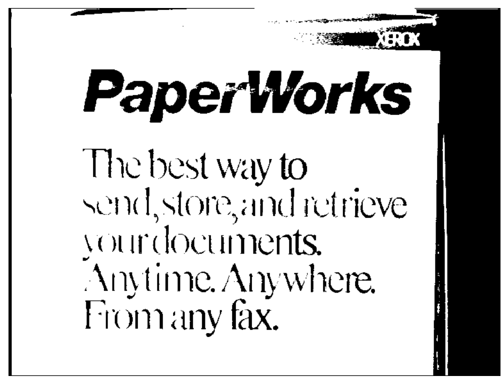


Figure 10b: approximate moving average scanning from right to left.

It would be good if the moving average did not work better with lighting from one direction or an other. Figure 11a shows the result of using another method to calculate the moving average. Instead of using the trailing line behind each pixel, it uses (an evenly weighted, not approximate) moving average that is centered about the n^{th} pixel. This uses a definition of $f(n)$ where

$$f_s(n) = \sum_{i=0}^{s-1} p_{n+\frac{s}{2}-i}$$

Using this method there are still some significant gaps in the letters, however, and it is also slower to calculate.

Figure 11a: centered moving average scanning from left to right.

Figure 11b: centered moving average scanning in alternate directions

Instead of traversing the image from left to right or right to left, another possibility is to traverse it alternatively from the left and from the right as illustrated in Figure 12. [This way of scanning was

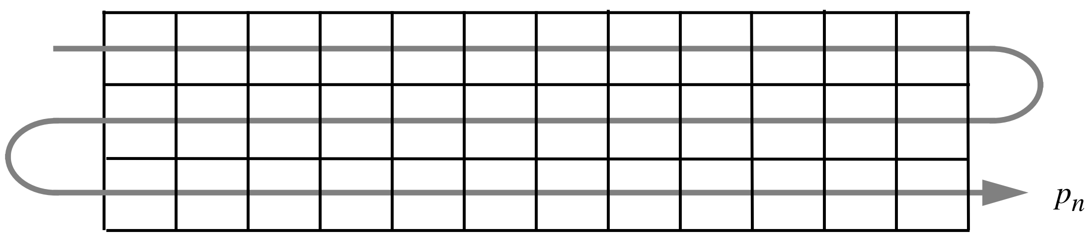


Figure 12: traversing pixels in alternate direction every other line.

called by the ancient Greeks *boustrophedon*, or "as the ox plows."] It makes very little difference with the centered average (as seen in Figure 11b), but if we go back to using the approximate moving average defined by $g(n)$, then alternate scanning improves things. The only anomaly is in the “gray” regions of an unevenly lit image, where going in one direction can produce the opposite result from going in the other, producing an every-other-line effect. (See Figure 13 which is

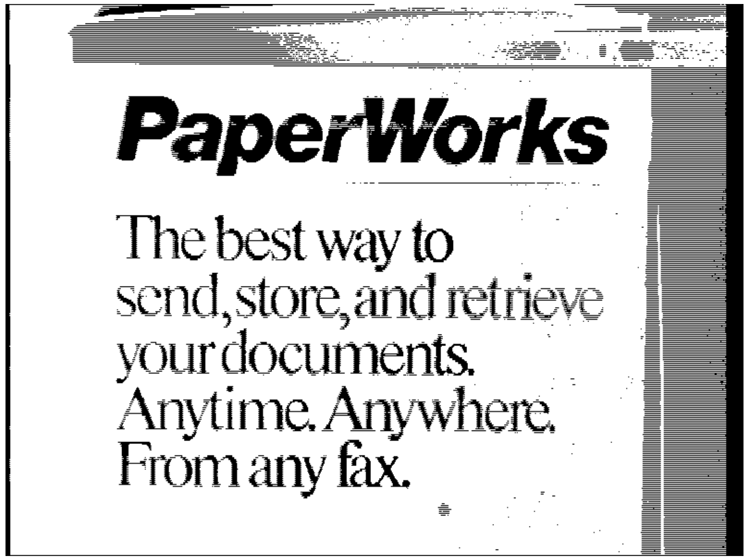


Figure 13: moving average alternatively from left to right and from right to left.

printed a little larger to show the effect better.) Superficially, this image may not look as pretty, but it is better than any of the previous algorithms in important ways because there are no big holes or parts of letters missing. The result is basically a combination of Figure 10a and Figure 10b, but the right-hand shadow has the every-other-line effect because when coming from the left, the background gets darker and the intensity of the shadow falls below the threshold, producing black. When the moving average wraps around the edge and includes the dark black edge (due to over-scanning), the threshold falls below the intensity of the shadow and the result is white as it traverses from right to left.

A small modification of this algorithm gets rid of the every-other-line effect and produces consistently good results across a wide range of images. The modification is to keep the previous line of approximate averages (which were calculated by scanning in the opposite direction) and to average the current line's average with the previous line's average, so we use

$$h(n) = \frac{g(n) + g(n - width)}{2}$$

This lets the threshold take some account of the vertical axis and produces the results in Figure 14.

Notice how well-formed all the letters are. Also, this is one of the few algorithms that does not

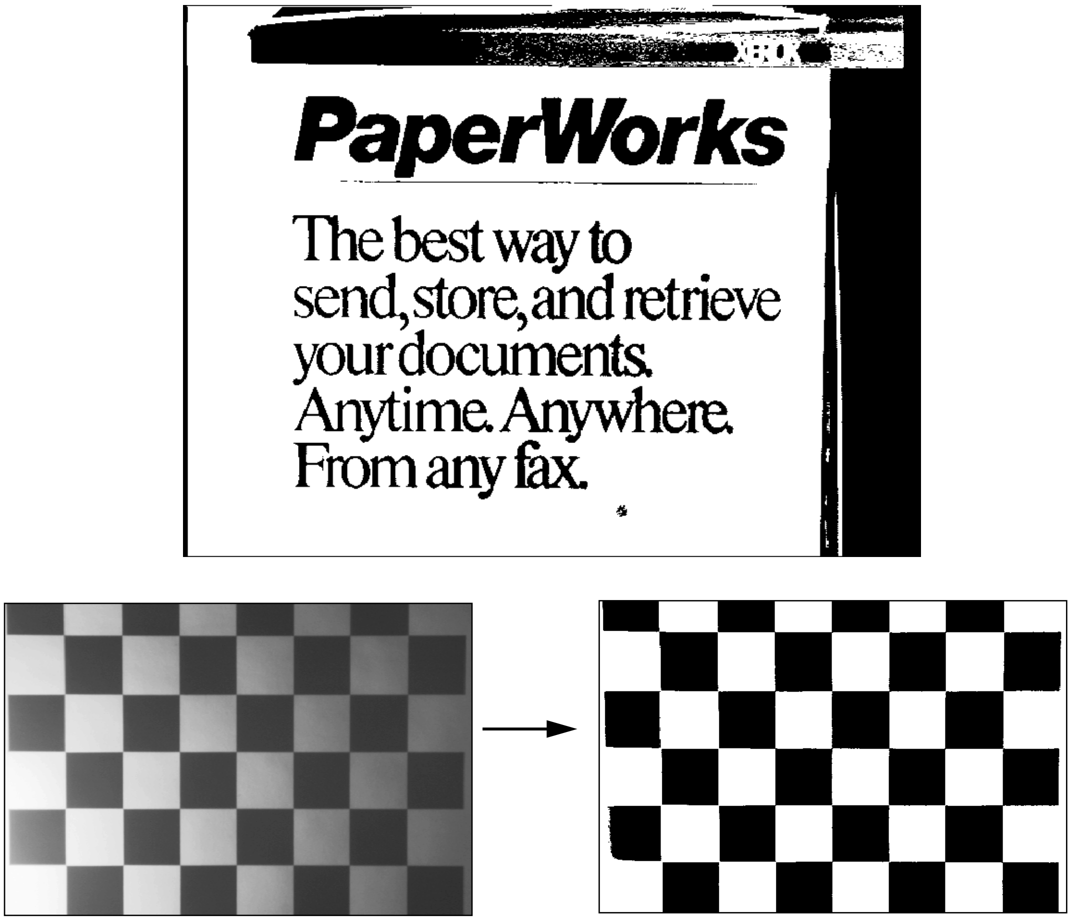


Figure 14: moving average alternatively from left to right and from right to left and averaging the two together.

eliminate the thin horizontal line below the word “PaperWorks.” Images whose intensity change mainly in the vertical direction are not as challenging as ones that change horizontally, but for the record, Figure 15 shows the result for the rotated image. It seems unlikely that further develop-

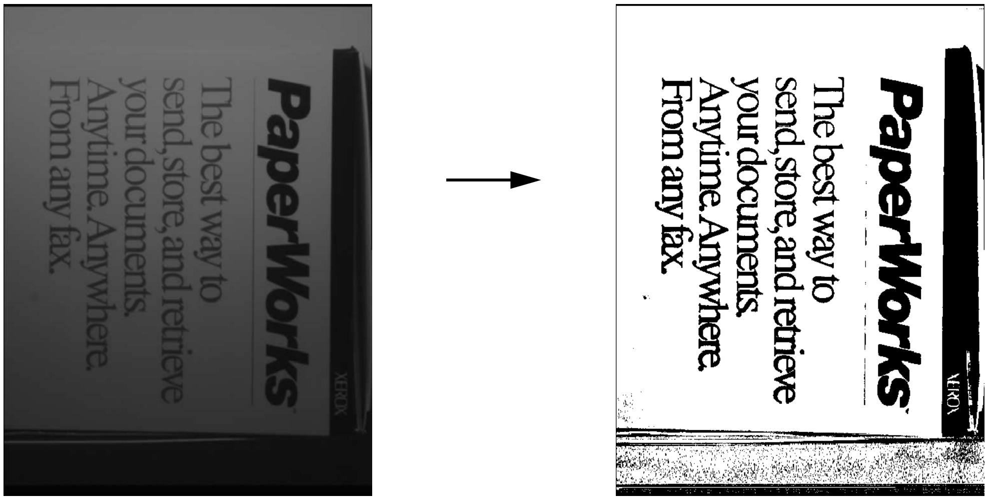


Figure 15: thresholding an image with vertical lighting variation

ments would produce a technique that generates significantly better results, so this is currently used for the DigitalDesk. The maximum amount of time it takes to run (for a full 768 x 575 image) is about 2 seconds on a SPARCstation 2 without paying special attention to optimization. Steve Freeman has modified this algorithm to go about 6 times faster by hard-coding multiplications and divisions to use powers of two so that shifting can be used instead. Further optimizations are possible, but will not make much difference for most interactions because they typically only operate on a small part of the image. For thresholded full-frame video as used by the DoubleDigitalDesk, and BrightBoard [6] however, these optimizations will be more important.

Acknowledgements

Thanks to Mike Flynn, Mik Lamming, Peter Robinson, Steve Freeman, and Quentin Stafford Fraser for helpful discussions on this topic.

References

1 Castleman, K. R. *Digital Image Processing*, Prentice-Hall, Englewood Cliffs, N.J. 1979.

2 Bartz, M. R. “The IBM 1975 Optical Page Reader. Part II: Video Thresholding System” IBM Journal of Research and Development, Sep. 1968, pp. 354-363.

3 Floyd, R. W. and Steinberg, L. An Adaptive Algorithm for Spatial Greyscale. *Proceedings of the S.I.D* 17, 2 (sedond quarter 1976), 75-77.

4 Haralick, Robert M. and Shapiro, Linda G. “Image Segmentation Techniques” Computer Vision, Graphics and Image Processing 29 January 1985, pp. 100-132.

5 Pratt, William K. *Digital Image Processing* John Wiley & Sons 1991.

6 Stafford-Fraser, Quentin "Brightboard..."

7 Weszka, Joan S. “A Survey of Threshold Selection Techniques” Computer Graphics and Image Processing 7, April 1978, pp.259-265

8 Standard reference to half-toning and dithering