# Adaptive Thresholding Using the Integral Image

Derek Bradley*
Carleton University, Canada
derek@derekbradley.ca

Gerhard Roth
National Research Council of Canada
Gerhard.Roth@nrc-cnrc.gc.ca

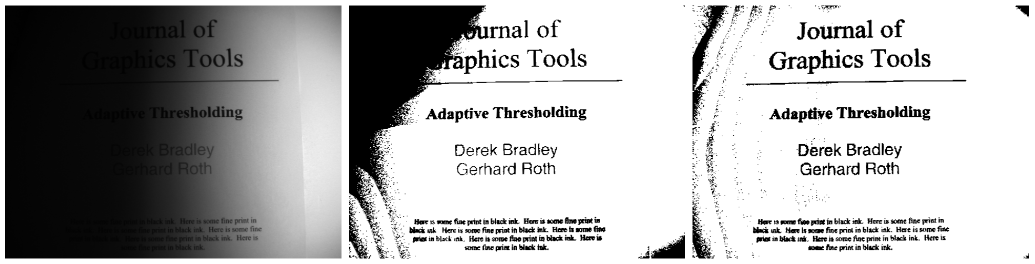
Journal of
Graphics Tools
Adaptive Thresholding
Derek Bradley
Gerhard Roth
Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink.

Figure 1: Real-time adaptive image thresholding. Left: Input image. Center: Wellner's previous technique. Right: Our technique.

**Abstract.** Image thresholding is a common task in many computer vision and graphics applications. The goal of thresholding an image is to classify pixels as either "dark" or "light". Adaptive thresholding is a form of thresholding that takes into account spatial variations in illumination. We present a technique for real-time adaptive thresholding using the integral image of the input. Our technique is an extension of a previous method. However, our solution is more robust to illumination changes in the image. Additionally, our method is simple and easy to implement. Our technique is suitable for processing live video streams at a real-time frame-rate, making it a valuable tool for interactive applications such as augmented reality. Source code is available online.

## 1 Introduction

Image thresholding segments a digital image based on a certain characteristic of the pixels (for example, intensity value). The goal is to create a binary representation of the image, classifying each pixel into one of two categories, such as "dark" or "light". This is a common task in many image processing applications, and some computer graphics applications. For example, it is often one of the first steps in marker-based augmented reality systems [Billinghurst et al. 2001; Bradley and Roth 2004; Fiala 2005], and it has been used in high dynamic range photography [Ward 2003]. The most basic thresholding method is to choose a fixed threshold value and compare each pixel to that value. These techniques have been described and evaluated extensively in a number of survey papers [Weszka and Rosenfeld 1978; Palumbo et al. 1986; Sahoo et al. 1988; Lee et al. 1990; Glasbey 1993; Trier and Jain 1995; Sezgin and Sankur 2004]. However, fixed thresholding often fails if the illumination varies spatially in the image or over time in a video stream.

In order to account for variations in illumination, the common solution is adaptive thresholding. The main difference here is that a different threshold value is computed for each pixel in the image. This technique provides more robustness to changes in illumination. A number of adaptive thresholding methods exist [White and Rohrer 1983; Bernsen 1986; Parker 1991; Wellner 1993; Yang et al. 1994; Shen and Ip 1997; Chan et al. 1998; Savakis 1998; Sauvola and Pietikainen 2000; Yang and Yan 2000]. Further examples and comparisons can be found in [Venkateswarlu and Boyle 1995; Sezgin and Sankur 2004]. We present a very simple and clear technique using integral images. Our

*Currently at The University of British Columbia, bradleyd@cs.ubc.ca method is easy to implement for real-time performance on a live video stream. Though our technique is an extension to a previous method [Wellner 1993], we increase robustness to strong illumination changes. In addition, we present a clear and tidy solution without increasing the complexity of the implementation. Our technique is also similar to the thresholding method of White and Rohrer for optical character recognition [White and Rohrer 1983], however we present an implementation designed for real-time video. The motivation for this work is finding fiducials in augmented reality applications. Pintaric also presents an adaptive thresholding algorithm specifically for augmented reality markers [Pintaric 2003], however his method requires that a fiducial has been located in a previous frame in order for the technique to threshold correctly. Our algorithm makes no assumptions and is more general, suitable for use in any application. The source code is available online at the address listed at the end of this paper.

## 2 Background

### 2.1 Real-Time Adaptive Thresholding

In this paper we focus on adaptively thresholding images from a live video stream. In order to maintain real-time performance, the thresholding algorithm must be limited to a small constant number of iterations through each image. Thresholding is often a sub-task that makes up part of a larger process. For instance in augmented reality, input images must be segmented to locate known markers in the scene that are used to dynamically establish the pose of the camera. A simple and fast adaptive thresholding technique is therefore an important tool.

### 2.2 Integral Images

An integral image (also known as a summed-area table) is a tool that can be used whenever we have a function from pixels to real numbers $f(x,y)$ (for instance, pixel intensity), and we wish to compute the sum of this function over a rectangular region of the image. Examples of where integral images have been applied include texture mapping [Crow 1984], face detection in images [Viola and Jones 2004], and stereo correspondence [Veksler 2003]. Without an integral image, the sum can be computed in linear time per rectangle by calculating the value of the function for each pixel individually. However, if we need to compute the sum over multiple overlapping rectangular windows, we can use an integral image and achieve a constant number of operations per rectangle with only a linear amount of preprocessing.

To compute the integral image, we store at each location, $I(x,y)$, the sum of all $f(x,y)$ terms to the left and above the pixel $(x,y)$. This is accomplished in linear time using the following equation for each pixel (taking into account the border cases),

$$I(x,y) = f(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1). \tag{1}$$

Figure 2 (left and center) illustrates the computation of an integral image. Once we have the integral image, the sum of the function for any rectangle with upper left corner $(x_1,y_1)$, and lower right corner $(x_2,y_2)$ can be computed in constant time using the following equation,

$$\sum_{x=x_1}^{x_2} \sum_{y=y_1}^{y_2} f(x,y) = I(x_2,y_2) - I(x_2,y_1-1) - I(x_1-1,y_2) + I(x_1-1,y_1-1). \tag{2}$$

Figure 2 (right) illustrates that computing the sum of $f(x,y)$ over the rectangle D using Equation 2 is equivalent to computing the sums over the rectangles (A+B+C+D)-(A+B)-(A+C)+A.

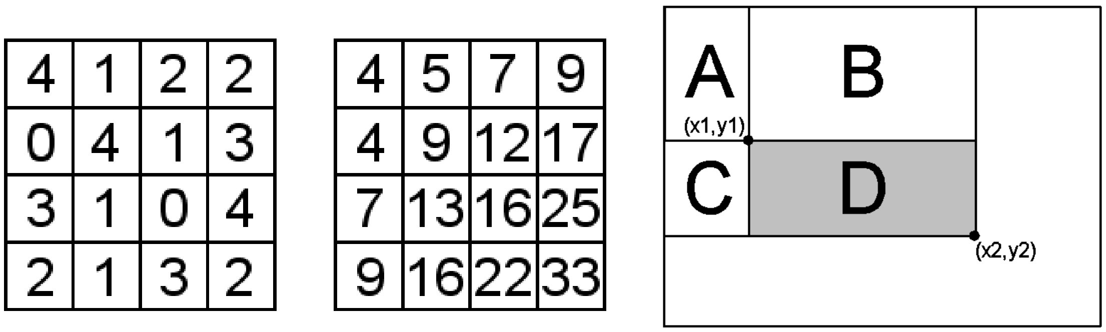

Figure 2: The integral image. Left: A simple input of image values. Center: The computed integral image. Right: Using the integral image to calculate the sum over rectangle D.

## 3 The Technique

Our adaptive thresholding technique is a simple extension of Wellner's method [Wellner 1993]. The main idea in Wellner's algorithm is that each pixel is compared to an average of the surrounding pixels. Specifically, an approximate moving average of the last *s* pixels seen is calculated while traversing the image. If the value of the current pixel is *t* percent lower than the average then it is set to black, otherwise it is set to white. This method works because comparing a pixel to the average of nearby pixels will preserve hard contrast lines and ignore soft gradient changes. The advantage of this method is that only a single pass through the image is required. Wellner uses 1/8th of the image width for the value of *s* and 15 for the value of *t*. However, a problem with this method is that it is dependent on the scanning order of the pixels. In addition, the moving average is not a good representation of the surrounding pixels at each step because the neighbourhood samples are not evenly distributed in all directions. By using the integral image (and sacrificing one additional iteration through the image), we present a solution that does not suffer from these problems. Our technique is clean, straightforward, easy to code, and produces the same output independently of how the image is processed. Instead of computing a running average of the last *s* pixels seen, we compute the average of an *s x s* window of pixels centered around each pixel. This is a better average for comparison since it considers neighbouring pixels on all sides. The average computation is accomplished in linear time by using the integral image. We calculate the integral image in the first pass through the input image. In a second pass, we compute the *s x s* average using the integral image for each pixel in constant time and then perform the comparison. If the value of the current pixel is *t* percent less than this average then it is set to black, otherwise it is set to white. The following pseudocode demonstrates our technique for input image *in*, output binary image *out*, image width *w* and image height *h*.

```
procedure $AdaptiveThreshold(in, out, w, h)$
1:  for $i = 0$ to $w$ do
2:      $sum \leftarrow 0$
3:      for $j = 0$ to $h$ do
4:          $sum \leftarrow sum + in[i, j]$
5:          if $i = 0$ then
6:              $intImg[i, j] \leftarrow sum$
7:          else
8:              $intImg[i, j] \leftarrow intImg[i-1, j] + sum$
9:          end if
10:     end for
11: end for
```

```
**for** $i = 0$ to $w$ **do**
  **for** $j = 0$ to $h$ **do**
    $x1 \leftarrow i - s/2$ {border checking is not shown}
    $x2 \leftarrow i + s/2$
    $y1 \leftarrow j - s/2$
    $y2 \leftarrow j + s/2$
    $count \leftarrow (x2 - x1) \times (y2 - y1)$
    $sum \leftarrow intImg[x2, y2] - intImg[x2, y1 - 1] - intImg[x1 - 1, y2] + intImg[x1 - 1, y1 - 1]$
    **if** $(in[i, j] \times count) \leq (sum \times (100 - t)/100)$ **then**
      $out[i, j] \leftarrow 0$
    **else**
      $out[i, j] \leftarrow 255$
    **end if**
  **end for**
**end for**
```

## 4 Performance and Examples

We tested our real-time adaptive thresholding technique on a Pentium 4 processor at 3.4Ghz with a Point Grey Dragonfly camera, capturing at a resolution of 640 x 480 pixels. On average our technique operates at approximately 15 milliseconds per frame, yielding a frame-rate of 65 frames per second. Since we require two iterations through each image rather than just one as in Wellner's method, we expect to perform slower. In practice we measured our technique at approximately 2.5 times slower than that of Wellner, however we still achieve a real-time frame-rate and a better segmentation.

Two examples of our adaptive thresholding result are presented in Figure 1 and Figure 3. The results of Wellner's method are also shown. Figure 1 illustrates a text example with a very dark shadow. Our method is able to segment all of the text in the image. Figure 3 illustrates an augmented reality example with a number of planar markers that are tracked in a video stream. In this example our technique provides near perfect segmentation despite the strong illumination changes in the image. Wellner's technique fails at the specular reflection in the center of the image at the bottom, and the dark shadow in the top corners.

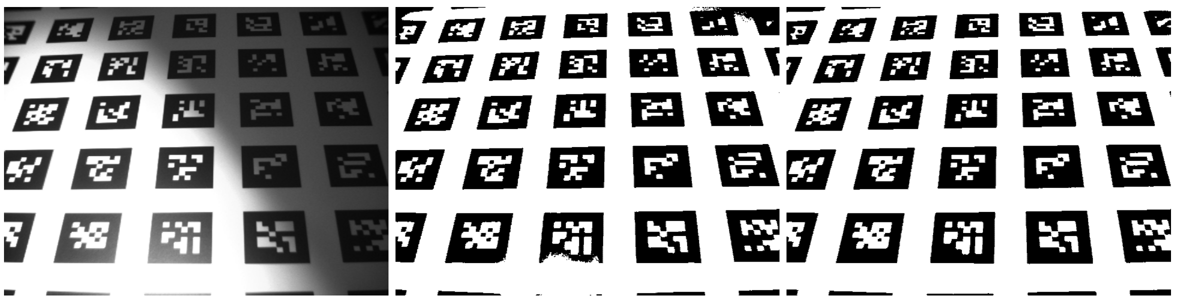
Figure 3: Augmented reality marker example. Left: Input image. Center: Wellner's method. Right: Our technique.

## 5 Discussion

Our previous work in augmented reality [Bradley and Roth 2004] demonstrates a need for robust adaptive thresholding of real-time video streams. Our technique is well-suited for scenes with strong spatial changes in illumination.

Temporal variations in illumination are also handled automatically, which is not the case for global thresholding methods. The main drawback of our method is that we must process the image twice. However, this is not a significant drawback with the current speed of processors and our technique is still suitable for real-time applications. The amount of processing can also be minimized by performing the thresholding step at pixel $(i-s/2, j-s/2)$ at the same time as computing the integral image at pixel $(i, j)$. In this case, a constant amount of processing must be performed for the following number of pixels,

$$(w \times h) + (\frac{s}{2} \times w) + (\frac{s}{2} \times h) - (\frac{s^2}{4}). \tag{3}$$

This technique for thresholding makes the assumption that the image contains primarily background pixels (to be segmented as white) and that the foreground pixels are rather distributed in the image. In the case of a foreground component that is larger than *s x s* pixels, the center of the component will be incorrectly classified as background. This problem can be alleviated by using a larger kernel (ie: by increasing *s*), however fine details in the segmentation may then be lost. The kernel size should be set appropriately according to the application.

## References

BERNSEN, J. 1986. Dynamic thresholding of gray-level images. In *Int. Conf. Pattern Recognition*, vol. 2, 1251–1255.

BILLINGHURST, M., KATO, H., AND POUPYREV, I. 2001. The magicbook - moving seamlessly between reality and virtuality. *IEEE Comput. Graph. Appl. 21*, 3, 6–8.

BRADLEY, D., AND ROTH, G. 2004. Augmenting non-rigid objects with realistic lighting. Tech. Rep. NRC 47398, National Research Council of Canada, October.

CHAN, F. H. Y., LAM, F. K., AND ZHU, H. 1998. Adaptive thresholding by variational method. *IEEE Transactions on Image Processing 7*, 3 (March), 468–473.

CROW, F. C. 1984. Summed-area tables for texture mapping. *SIGGRAPH Comput. Graph. 18*, 3, 207–212.

FIALA, M. 2005. ARTag, a fiducial marker system using digital techniques. In *Proc. of Computer Vision and Pattern Recognition*, vol. 2, 590–596.

GLASBEY, C. A. 1993. An analysis of histogram-based thresholding algorithms. *CVGIP: Graph. Models Image Process. 55*, 6, 532–537.

LEE, S., CHUNG, S., AND PARK, R. 1990. A comparative performance study of several global thresholding techniques for segmentation. *Computer Vision Graphics Image Processing 52*, 2 (Nov.), 171–190.

PALUMBO, P. W., SWAMINATHAN, P., AND SRIHARI, S. N. 1986. Document image binarization: Evaluation of algorithms. *SPIE 697*, 278–285.

PARKER, J. R. 1991. Gray level thresholding in badly illuminated images. *IEEE Trans. Pattern Anal. Mach. Intell. 13*, 8, 813–819.

PINTARIC, T. 2003. An adaptive thresholding algorithm for the augmented reality toolkit. In *IEEE Int. Augmented Reality Toolkit Workshop*.

SAHOO, P. K., SOLTANI, S., WONG, A. K., AND CHEN, Y. C. 1988. A survey of thresholding techniques. *Comput. Vision Graph. Image Process. 41*, 2, 233–260.

SAUVOLA, J., AND PIETIKAINEN, M. 2000. Adaptive document image binarization. *PR 33*, 2 (February), 225–236.

SAVAKIS, A. E. 1998. Adaptive document image thresholding using foreground and background clustering. In *ICIP (3)*, 785–789.

SEZGIN, M., AND SANKUR, B. 2004. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging 13*, 1, 146–168.

SHEN, D., AND IP, H. H. S. 1997. A hopfield neural network for adaptive image segmentation: An active surface paradigm. *PRL 18*, 37–48.

TRIER, O. D., AND JAIN, A. K. 1995. Goal-directed evaluation of binarization methods. *PAMI 17*, 12 (December), 1191–1201.

VEKSLER, O. 2003. Fast variable window for stereo correspondence using integral images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, 556–661.

VENKATESWARLU, N. B., AND BOYLE, R. D. 1995. New segmentation techniques for document image analysis. *IVC 13*, 7 (September), 573–583.

VIOLA, P., AND JONES, M. J. 2004. Robust real-time face detection. *Int. J. Comput. Vision 57*, 2, 137–154.

WARD, G. 2003. Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures. *journal of graphics tools 8*, 2, 17–30.

WELLNER, P. D. 1993. Adaptive thresholding for the digitaldesk. Tech. Rep. EPC-93-110, EuroPARC.

WESZKA, J. S., AND ROSENFELD, A. 1978. Threshold evaluation techniques. *IEEE Trans. on System, Man and Cybernetics SMC-8*, 8, 622–629.

WHITE, J. M., AND ROHRER, G. D. 1983. Image thresholding for optical character recognition and other applications requiring character image extraction. *IBMRD 27*, 4 (July), 400–411.

YANG, Y., AND YAN, H. 2000. An adaptive logical method for binarization of degraded document images. *PR 33*, 5 (May), 787–807.

YANG, J. D., CHEN, Y. S., AND HSU, W. H. 1994. Adaptive thresholding algorithm and its hardware implementation. *PRL 15*, 141–150.

## Web Information:

Source code for our technique and additional image examples are available on the web. The web address is www.derekbradley.ca/AdaptiveThresholding/index.html.