2011 International Conference on Document Analysis and Recognition

Detection and Segmentation of Antialiased Text in Screen Images

Sivan Gleichman[1], Boaz Ophir[2], Amir Geva[1], Mattias Marder[1], Ella Barkan[1], Eli Packer[1]
[1]*IBM Research - Haifa, Israel*
[2]*Computer Science Department, Technion - Israel Institute of Technology*

***Abstract*—Various software applications deal with analyzing the textual content of screen captures. Interpreting these images as text poses several challenges, relative to images traditionally handled by optical character recognition (OCR) engines. One such challenge is caused by text antialiasing, a technique which blurs the edges of characters, to reduce jagged appearance. This blurring changes the character images according to context, and can sometimes fuse them together. In this paper, we offer a low-cost method that can be used as a preprocessing stage, prior to OCR. Our method locates antialiased text in a screen image and segments it into separate character images. Our proposed algorithm significantly improves OCR results, particularly in images with colored text of small font size, such as in graphic user interface (GUI) screens.**

***Keywords*-antialiasing; character segmentation; text detection;**

I. Introduction

In many applications it is essential to recognize the content of computer screens. Translation is the most common example. However, there are various additional business domains where automatic operation of the user interface is of interest. This automation requires the analysis of the user interface screens [1]. Such services include: automation of business processes that have traditionally been performed by back office human operators; automatic access to legacy systems for various purposes, such as data migration; automatic visual software testing; and automatic monitoring of human operators activities, etc.

In general, the automation of these processes can also be performed without analyzing the user interface screens. That is, by accessing the system's internal structure. However, in this approach a state-of-the-art software must interface with legacy systems via a unique protocol (API) or by instrumenting it, i.e., by modifying or inserting "spy code". This approach is problematic for various reasons. It requires knowledge of the legacy system's internals and the development of a system interface. Moreover, some legacy systems are not accessible for editing, and may not have APIs. In addition, instrumenting the system may affect its behavior in unintended ways, possibly altering the results of the process.

A visual approach, which directly access an application through its user interface, can improve upon this problematic approach. Furthermore, a visual approach enables operation using remote desktop protocols, in which only the screen image is available. In this approach, an application's screen is captured and analyzed to recognize the state of the user interface. This analysis usually involves both image processing and OCR. The image processing goal is to recognize graphical features, such as lines and rectangles, in order to deduce the screen layout. The OCR's role is to extract text such as titles and text fields.

Screen images are acquired under seemingly ideal conditions, with almost no noise, thereby appearing simple to analyze. However, interpreting screen images poses several challenges. One such challenge is caused by text antialiasing (AA), which involves horizontal blurring of the character images. While AA tends to improve text readability for the human user [7], it causes severe problems for OCR systems. Especially in small font sizes, AA characters tend to touch one another, making contour- and projection-based segmentation methods inapplicable.

This paper addresses methods for locating AA text within screen images and correctly segmenting the characters. We focus on sub-pixel AA, which is widely used in both Windows an Linux platforms. Our only assumptions are that characters in a single word are all of a single color, and that the immediate background of the word is quite uniform. These assumptions are valid in vest majority of practical applications. Our goal is to reach good segmentation results with very low latency.

Different approaches to handling AA small font sizes have been suggested in previous work. In [2]–[4], a joint segmentation and recognition approach is used. First, word images are segmented (and over-segmented), and then the segments are merged and classified. This requires many classification operations (more than the number of actual characters), and as such it is too expensive to be practical for low latency needs. In [5], [6], a Hidden Markov Model (HMM) is used for joint segmentation and recognition. An HMM model is matched to features extracted from a sliding window over the word image. Since this approach requires the computation of HMM models for characters of all font types and styles it is also too expensive to be practical for our needs.

Contrary to the works just mentioned, our approach utilizes the understanding of the process of AA text image creation. We propose a low-cost procedure that can be used as a preprocessing step prior to OCR. Usually the performance of OCR engines deteriorates when operating on images with only a few words and with small font sizes. Moreover, when the text and background colors share common RGB channels, such as blue text on black background, OCR results are usually poor. As we demonstrate in this paper, using our proposed method prior to OCR considerably improves the results for these cases.

This paper is organized as follows. In Section II, we describe the process of AA text image creation. We explain our detection and segmentation algorithms in Section III, and we present their results in Section IV. Section V

1520-5363/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDAR.2011.92

concludes the work.

II. Antialiased Text Image Creation

We start by explaining the conceptual steps for rendering an AA text image. In practice, the rendering process, such as [7]–[9], may not follow these exact steps. However, these steps represent the concept behind the process, and explain the graphical characteristics of AA text images. In the first step, the program generating the screen image computes the characters' positions. Next, an AA filter is applied on an image with black text on a white background, according to the computed characters' positions. Finally, text and background colors are applied.

Typical AA filters are horizontal low-pass filters. They can be applied at full pixel or sub-pixel resolution, according to system and program settings. In this work, we focus on sub-pixel AA filtering, such as Microsoft ClearType™ [7], [8], which is one of the most common sub-pixel AA techniques in use today. Figure 1 is an example of black text on a white background, with and without ClearType AA.

(a) No antialiasing (b) Cleartype antialiasing

Figure 1. Black on white antialiasing example

Due to the AA process, a character image is affected by the neighboring characters. In particular, as shown in Figure 1, AA text may not contain background pixels separating adjacent characters. As a result state-of-the-art segmentation methods, used in classical OCR systems [10], may fail.

Color is applied to the rendered characters using the following formula:

$$y = T_c + \frac{B_c - T_c}{255}x \quad , c \in \{r, g, b\} \tag{1}$$

in which y is the final sub-pixel intensity, x is the sub-pixel intensity after the AA filtration and before the coloring, T is the text color, and B the background color. The different color channels do not interact, so the intensity of each sub-pixel is affected only by the text and background color components that are associated with the sub-pixel position. Adjacent sub-pixels are thus affected differently by the coloring. Some examples for colored AA text are presented in Figure 2.

(a) Red on white (b) Yellow on blue

Figure 2. Colored AA text examples

III. Antialiased Text Segmentation

The idea of our proposed AA text segmentation process is to reverse the AA text creation precess. First, we detect the location of the AA text. Then, for each detected area we invert the color transformation in (1) and create a grayscale, sub-pixel image. Finally, we perform character segmentation, which is deconvolution of the AA filtration, resulting in a binary black on white text image. The connected components of this binary image are the desired characters. Figure 3 summarizes the process. We describe each of its steps in detail below.

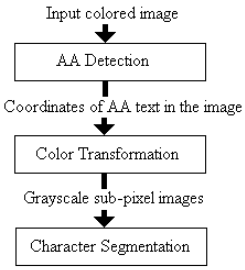


Figure 3. Antialiased text segmentation - process scheme

A. Antialiased Text Detection

In this step, we identify the location of AA text on the screen. Many different text detection schemes have been proposed for different scenarios [11]. However, none of the previous works address our specific problem. Furthermore, such solutions do not differentiate between AA and non-AA text, whereas some rendering technologies (e.g., Flex) allow mixing different types of text in the same screen.

Our AA text detector is based on two stages: first, locating image areas with horizontal gradient and then checking if the histogram properties of these areas correspond AA text. To detect gradient areas, we transform the image into grayscale, whereas the histogram test is preformed on each color channel separately.

1) Gradient detection: A positive horizontal gradient indicator D_p is defined on a grayscale image I by:

$$D_{p_{i,j}} = \begin{cases} 1 & , \quad I_{i,j-1} + \Delta < I_{i,j} < I_{i,j+1} - \Delta \\ 0 & , \quad otherwise \end{cases} \tag{2}$$

This indicates pixels that are lighter than their neighbor to the left and darker than their neighbor to the right, by the predetermined threshold Δ. A negative horizontal gradient indicator D_n is similarly defined, and a general horizontal gradient indicator is defined by $D = D_p \cup D_n$. An example of these indicators is presented in Figure 4. The black pixels in image (b) are those indicated by D_n, and the gray are those indicated by D_p.

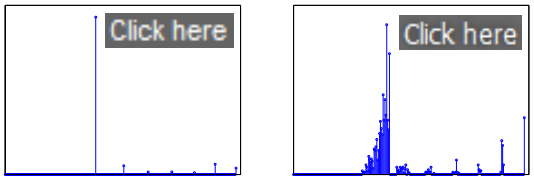
(a) Input image (b) D_p (gray) and D_n (black)

Figure 4. Gradient indicators

Connected components of the general gradient D are then grouped into equivalence classes according to proximity. In order to eliminate non-text gradient areas, equivalence classes with either too few pixels or too large bounding boxes are ignored. Furthermore, since we expect gradients of opposite orientation on both sides of each character, we also dismiss equivalence classes with only one type of gradient. In general the threshold parameters for the classification and elimination of equivalence classes

are set according to the expected font size. However, as can be seen in our simulations in Section IV, this setting is robust and fits a large range of font sizes. We refer to the bounding boxes of the remaining equivalence classes as AA candidate areas, and test their histograms for AA text properties.

2) Histogram test: In order to determine whether a candidate area contains AA text, we test its histogram in each color channel. It is easy to see that the histogram of AA text consists of a few (usually about seven) nonzero bins, which are spread quite uniformly between the background and text colors. When the background color is not entirely uniform, instead of isolated nonzero bins, the histogram consists of small groups of sequential nonzero bins. Figure 5 introduces two typical histograms of AA text, on uniform background and on approximately uniform background. The histograms in this figure are of the red channel of the presented text images.



(a) Uniform background (b) Approximately uniform

Figure 5. Histogram examples - red channel only

We divide the histogram into groups of sequential nonzero bins. If this results in too many such groups, or if they consists of to many bins, as compared to preset thresholds, then the candidate area is declared as not AA text. Moreover, in a histogram of AA text, the group of sequential bins with the largest bin's sum usually corresponds to the background color, which is one of the outermost nonzero bins. Therefore, we can also eliminate candidate areas according to the location of the group with the largest sum. However, in some noisy cases this claim could be inexact; therefore, we use it only when we know a priori that the image is of high color quality.

Figure 6 presents our AA detector performance on a screen from a well known application. The detected areas are marked by red rectangles. As shown in figure 6, all AA text areas are detected, and only very few non-text areas are falsely detected. These false detected areas are the arrows in the navigation toolbar, which are very similar to AA text. Even though the background of the buttons and tab caption is not uniform, the AA text is still detected.

B. Color Transformation

In order to invert the color transformation in (1), we first need to find the background and text colors and tolerances. For that we look at the histogram of each color channel, and calculate the sum of each three sequential bins. We decrease the histogram resolution and recalculate the sums, until the largest sum contains one of the exterior nonzero bins. We set the background color according to this exterior nonzero bin, and the text color according to

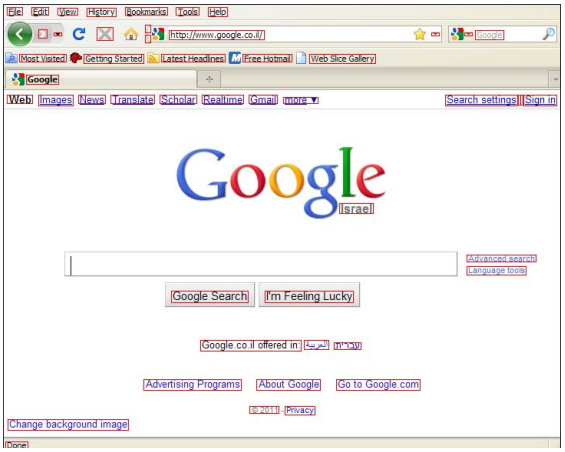


Figure 6. Antialiased text detection

the opposite exterior nonzero bin. We define the text color tolerance according to the final histogram resolution. The background color tolerance is set relative to the number of nonzero bins surrounding the background color in the histogram.

We now invert (1) to recover the sub-pixel grayscale image and achieve decoloring. For this process, we refer to three types of pixels in the colored image: background pixels, text pixels, and intermediate pixels. A background pixel is a pixel whose color is close to the background color, in all channels, relative to the background color tolerance. Similarly, A text pixel is defined according to the proximity to the text color, relative to the text color tolerance. The rest of the pixel are intermediate. We recover the sub-pixel grayscale image as follows: sub-pixels corresponding to background pixels in the colored image, are set to 255; sub-pixels corresponding to text pixels are set to 0; and sub-pixels corresponding to intermediate pixels are set according to the inverted formula:

$$x = \frac{255}{B_c - T_c}(y - T_c) \quad , c \in \{r, g, b\} \tag{3}$$

in which x is the sub-pixel intensity in the decolored image, and y is the intensity of the synonymous sub-pixel in the colored image.

Exceptional cases occur when one or two color components are identical in both text and background. In those cases, (3) is irrelevant. When there is only one such degenerate channel, the values of the corresponding sub-pixels are set to the average of their immediate horizontal neighbors. In case of two degenerate channels, the sub-pixel resolution no longer adds any value. Therefore, the decolored image is set according to the non-degenerate channel alone.

Care must be taken with the spatial order of the color channels (RGB or BGR). This order can be determined by the direction of the gradients in the sub-pixel image. We expect negative gradient on the left of each character and positive on the right. We build the sub-pixel image according to RGB, therefore if the order of the gradients in the sub-pixel image is reversed, then we conclude that

the original order was BGR and rearrange the sub-pixels accordingly.

In Figure 7, we present three examples of decoloring. On the top row, the background color is not uniform. In the middle row, the text and background colors share the same blue component, and on the bottom row, both the blue and the green components are shared. Our algorithm performs very well on all three examples.

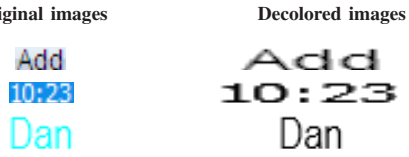


Figure 7. Decoloring examples

C. Character Segmentation

At this point, we have a grayscale image of AA black text on white background, and we can finally turn to character segmentation. Our character segmentation approach is in fact deconvolution of the AA filtration. This deconvolution results in a binary black on white text image, whose connected components correspond to the desired characters. We refer to this binary image as *the mask*.

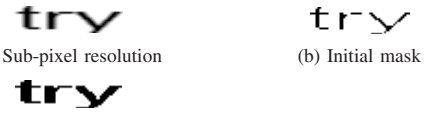
Standard deconvolution approaches, such as Weiner filtering or more complex deconvolution schemes involving optimization of energy-functional, produce insufficient results. These results are especially poor in small fonts or when the text and background share one or two common color components. Instead we suggest a very efficient deconvolution approach. In this approach, we grow the mask according to horizontal gradients in the image, as explained below. This method can merge segments in adjacent rows. In most cases this merge is desirable, since the segments belong to the same character, but in some cases characters can merge by mistake. However, we can identify these cases, since the connection between the characters is usually weak. In the segmentation algorithm, we mark such suspicious connections as potential breakpoints. Later, if the OCR does not recognize a whole segment, we break it at the potential breakpoint and recognize each partial segment separately.

The character segmentation algorithm consists of 5 stages:

1) Creating initial mask
2) Growing the mask
3) Finding potential breakpoints
4) Filling holes in the mask
5) Segmenting according to the mask's connected components

In stage 1, we initiate the mask to indicate both pixels, which are below a threshold and local minima. We also look for local maxima which are potential connections between characters. If the gray level of a local maximum is above a preset threshold, we conclude that it is a connection point between characters, and set it to the background level of 255. Otherwise, we assume that it is inside a character and add it to the mask. However, in the latter case, the local maximum can still be a connecting point, so if the values of its surrounding pixels are not low enough, we mark it as a potential breakpoint. We will identify the rest of the potential breakpoints in stage 3.

In stage 2, we scan the image and mask twice: a left-to-right scan and a right-to-left scan. Each pixel in the scan is added to the mask only if it fulfills the following requirements: its value is below a preset threshold; only one of its immediate neighbors is on the mask; and it is on a gradient, i.e., it is brighter or equal to its masked neighbor and darker than its opposite neighbor. An example of the evaluation in stage 2 is illustrated in Figure 8. Even though the characters 'r' and 'y' are connected by bright pixels in the sub-pixel image, they are correctly separated in the mask.



(a) Sub-pixel resolution (b) Initial mask

(c) After mask growing

Figure 8. Mask growing

As shown in Figure 8, generally at this point, each connected component in the mask represents a character. However, in some cases, character masks can be connected, therefore in stage 3 we mark potential breakpoints. Figure 9 presents mask connections that we look for in stage 3. If we find such a connection, and if its surrounding on the sub-pixel image does not contain enough dark pixels, then we mark it as a candidate breakpoint. In the case of diagonal connection, as in the images on the left of Figure 9, we also add pixels to the mask, so that this connection holds when using four neighbors connectivity.

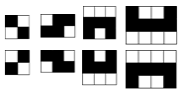


Figure 9. Mask connection types

Figure 10 presents an example for a potential breakpoint and a connection of characters on the mask. The candidate breakpoint is the red pixel in images (c) and (d). As shown in the letter 'A' in image (c), holes can occur in the mask-growing process. In stage 4, we fill such holes to achieve smooth character images. That is, we add to the mask all pixels which have at least 7 out of 8 masked neighbors.

(a) Input image (b) Sub-pixel image

(c) After mask growing (d) After holes filling

Figure 10. Candidate breakpoint - in red

In stage 5, we complete the character segmentation by dividing the mask into connected components, using four neighbors connectivity. Each of these components corresponds to a character or connected characters, usually with breakpoints. However, even without AA, in some

combinations of characters, font, and font size, characters are fused together with no background pixels separating them. In these cases, our algorithm cannot separate the characters, and expensive joint segmentation-recognition approaches must be applied.

IV. Results

We first present the performance of our process on a benchmark of real screen images, such as the one presented in Figure 6 in Section III-A. These images are screen captures taken from various applications both in Windows and Linux, and contain AA text in various colors, fonts, and font sizes. For all screens we used the same parameters setting. The gradient threshold was $\Delta = 5$. The proximity threshold for the gradient equivalence classes was 2, and the minimal number of pixels in an equivalence class was 6. The maximal number of sequential nonzero groups in the histograms was 11, and their maximal length was 35. The threshold for initiating the mask for character segmentation was 70.

The total number of AA words in the benchmark was 628, out of which our detector successfully detected 99.2%. Our algorithm falsely detected a total of 35 non-text areas in all images. The total number of character segments in all the detected words was 3355, out of which 99.8% were successfully segmented in the character segmentation step. A character segment can contain more than one character, since our segmentation cannot separate characters that are connected even before the AA filtering.

Next, we assess the quality of our method by testing the improvement it yields in character recognition, using a leading OCR engine. We created synthetic images with colored AA text, similar to Figure 11, with different colors and font sizes. We used the font 'Arial' for all the images, and the font size varied from 10 to 14.

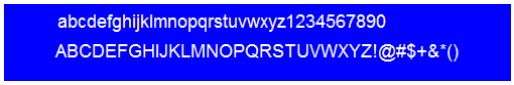


Figure 11. Example of a colored image

First we operated the OCR engine on these synthetic images. These images are very difficult for OCR engines, due to the short text, small size, and colors. Therefore it produced extremely poor results. Next, we applied our method on the synthetic images, using the same parameters as before, and created new images with the segmented characters in black on white. The OCR engine was operated again on the new images. Table I presents the OCR success rates, with and without our preprocessing, for each color, averaged over the font sizes. In all cases, our preprocessing significantly improved the results.

We also performed the same process on images with much longer text of a bit larger font size, and when the text and background do not share common color channels. In these cases the OCR success rate was close to 100% both with and without our preprocessing. That is, our preprocessing did not deteriorate the OCR results when they were good, and significantly improved them in the difficult cases.

	Success Rate	
	Without Preprocessing	With Preprocessing
color 1	17.5%	94.6%
color 2	53.8%	92.7%
color 3	70.1%	91.3%
color 4	0.0%	92.7%
color 5	25.6%	94.4%
color 6	58.9%	93.2%
color 7	57.2%	93.0%
color 8	64.8%	95.5%

Table I
OCR success rate for very difficult cases with and without preprocessing

V. Conclusion

We introduced a novel, low-cost procedure for the detection and segmentation of sub-pixel AA text on screen images. Using this procedure as a preprocessing stage prior to OCR has low overhead and can extremely improve OCR results, especially in cases of short and colored text with small font size, such as GUI screens. Moreover, our method is not limited to a specific platform, providing good results on several platforms that use sub-pixel AA, such as Windows and Linux. We are currently integrating the proposed method with a fast OCR engine optimized for screen images. Our future work will focus on adding an adaptive element to the OCR engine, designed to handle segmentation errors and connected characters.

References

[1] A. Geva and E. Walach, "US 2008/0001959 A1: System, method and computer program product for performing information transfer using a virtual operator," US Patent Application, Jan. 2008.

[2] S. Wachenfeld, H. Klein, and X. Jiang, "Recognition of screen-rended text," in *Proc. of the 18th Int. Conf. on Pattern Recognition*, vol. 2, 2006, pp. 1086–1089.

[3] S. Wachenfeld, H. Klein, S. Fleischer, and X. Jiang, "Segmentation of very low resolution screen-rendered text," in *Proc. of Int. Conf. on Doc. Anal. and Rec.*, 2007.

[4] S. Wachenfeld, S. Fleischer, and X. Jiang, "A multiple classifier approach for the recognition of screen-rendered text," in *Proc. of Int. Conf. on Computer Analysis of Images and Patterns*, 2007, pp. 921–928.

[5] F. Einsele, R. Ingold, and J. Hennebert, "A hmm-based approach to recognize ultra low resolution anti-aliased words," in *Int. Conf. on Pat. Rec. and Mach. Int.*, 2007.

[6] F. Einsele and R. Ingold and J. Hennebert, "A language-independent, open-vocabulary system based on hmms for recognition of ultra low resolution words," *Journal of Universal Computer Science*, vol. 14, no. 18, pp. 2982–2997, 2008.

[7] J. Sheedya, Y. Taia, M. Subbaramb, S. Gowrisankaranc and J. Hayes, "ClearType sub-pixel text rendering: Preference, legibility," *Displays, Elsevier*, vol. 29, issue. 2, pp. 138–151, 2008.

[8] C. Betrisey *et al.*, "Displaced filtering for patterned displays," in *Proc. Soc. for Inf. Display Symposium*, 2000, pp. 296–299.

[9] J. Platt, "Optimal filtering for patterned displays," *IEEE Signal Processing Letters*, vol. 7, no. 7, pp. 179–181, 2000.

[10] G. Nagy, "Twenty years of document image analysis in PAMI," *IEEE Trans. on Pat. Anal. and Machine Int.*, vol. 22, no. 1, pp. 38–62, 2000.

[11] K. Jung, K. Kim, and A. Jain, "Text information extraction in images and video: a survey," *Pattern Recognition*, vol. 37, no. 5, pp. 977–997, 2004.