Undoing Subpixel Rendering for Better Screenshots

J. Loviscach

Fachhochschule Bielefeld (University of Applied Sciences), Bielefeld, Germany

Abstract

Subpixel rendering employs the spatial layout of the red, green, and blue slices of a single square pixel on a screen to enhance the horizontal resolution. This method has become the de-facto standard for displaying text. Printed screenshots and screen magnifier software, however, reveal color seams. This is particularly vexing when screenshots are to be printed in color at a larger-than-normal scale, for instance in a tutorial article in a computing magazine. Hence, this work introduces a method for automatic correction.

Categories and Subject Descriptors (according to ACM CCS): I.4.3 [Image Processing and Computer Vision]: Enhancement—Filtering

1. Introduction

These days, most of the texts on color computer displays are rendered addressing not each pixel as a whole but leveraging the geometric distribution of the red, green, and blue subpixels to enhance the horizontal resolution, see Figure 1. This idea forms the basis of Microsoft ClearType [BBD*00, GTAE04] and Adobe CoolType and has also been implemented in Apple's graphics library Quartz. A regular screenshot, however, does not capture the geometric arrangement of the subpixels. Rather, it is thought to be composed of square pixels that are fully covered by each of the red, green, and blue components. This mismatch leads to color seams when magnifying a screenshot—for instance by using screen magnifier software—or when printing it, whereas the correct image should be grayscale, see Figure 1.

Redoing screenshots with subpixel rendering switched off is hardly an option in a busy editorial office. Here, an automated correction would be appreciated. This is an ill-posed problem as there is not enough information in the screenshot to recover the "real" image: A color seam may stem from subpixel rendering, but it may, for instance, also be an original part of an icon or a photo. Classically rendered items—including text in legacy software—and subpixel renderings appear side by side on the screen. A final twist is that text may be colored (which this work ignores) and/or be placed on a colored background (which this work handles).

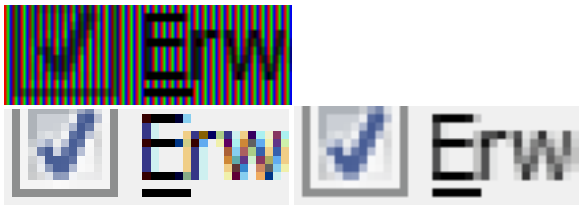

Figure 1: *RGB subpixel rendering makes use of the layout of the light-emitting areas on a computer screen (top). This leads to artificial color seams in regular screenshots (bottom left), in contrast to a correct representation (bottom right).*

2. Method

Given the ill-posedness of the problem, the proposed method resorts to machine learning, even though in a basic form. It builds a model for what constitutes text rendered via subpixels and then classifies each pixel in a screenshot in a binary fashion: Is it a potential color seam of text or is it not? This step is related to text detection methods such as [YJHY07]. If a pixel is classified as belonging to text, its color is corrected.

The basis for deciding which pixels constitute text is a screenshot of about 600×700 pixels showing about 6,000 characters in different fonts, styles, and sizes rendered with ClearType. To achieve a compact data model, all horizontal sequences of three pixels are extracted as nine-dimensional data vectors $(R_1 G_1 B_1 R_2 G_2 B_2 R_3 G_3 B_3)$. This initial list re-

© The Eurographics Association 2010.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

duced to a shorter list that only contains data vectors that are not almost grayscale and that have a pairwise Euclidean distance of more than 0.1. (This can be raised to 0.3 with only little detrimental effect.) This yields a collection of about 2000 nine-dimensional data vectors, to be used as a model of text. The processing up to here can be done once for all and be reused for all screenshots to be processed.

Every horizontal group of three pixels in the screenshot to be processed is compared to the collection of data vectors in the model. Upfront, to cater for a potentially non-white background, the three RGB values of the screenshot are scaled up by $\frac{1.1}{0.1+\max}$, where max is the maximum value in the R, G, or B channel, respectively, in a 5×5 neighborhood. If the Euclidean distance of this nine-dimensional data vector to some element of the data model is less than 0.2, its position is marked in a map. (A value of 0.3 instead of 0.2 increases the number of false positives dramatically.)

As there may be isolated spurious detections, these marks are filtered in a final step: Every pixel whose 5×5 neighborhood contains six or more marks is considered text. These are the pixels whose color is corrected toward grayscale. To take into account that the background color may not be white, this grayscale value is tinted by multiplying its R component with the maximum R value in a 5×5 neighborhood; similarly for the G and B component. The 5×5 neighborhoods mentioned in this paragraph and the one before cannot reliably be replaced by 3×3 neighborhoods.

3. Results and Conclusion

This method has been implemented in MATLAB® and has been tested successfully on a variety of screenshots. Figure 2 shows that text on colored backgrounds is detected—as partially also is light text, even though it is not covered explicitly by the method. Figure 3 shows a case that is particularly tricky, due to complex hard-edged icons. In addition to some minor spots in other icons, the yellow halo of the magic wand tool is misclassified as text. As the color correction tints the pixels with the background color, such false positives do not lead to noticeable errors, which suggests a certain resilience of the method. False negatives, on the other hand, tend to only occur with colored or light gray text.

This work introduced an unsupervised method to correct for the color seams generated by printing and/or magnifying text rendered into RGB subpixels. The method is based on a codebook gained from examples. It does not yet incorporate examples of what does *not* constitute pixels to be corrected. The next step is an implementation of the method as a plug-in to a standard image manipulation software. To allow the user to correct errors introduced in complex situations, there may be two plug-ins: one to create the mask (see right part of Figure 2), and one plug-in to correct the colors of the selected parts of an image toward a multiple of the local background color.

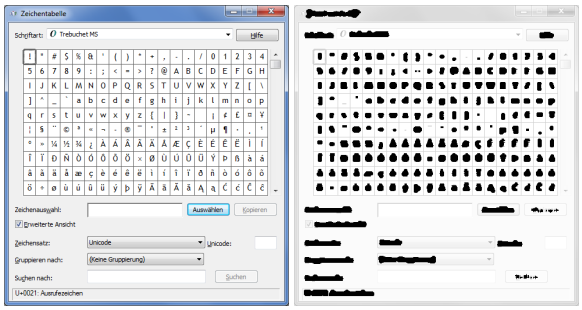

Figure 2: *Processing a screenshot of the Microsoft Windows character table (left: original; right: mask).*

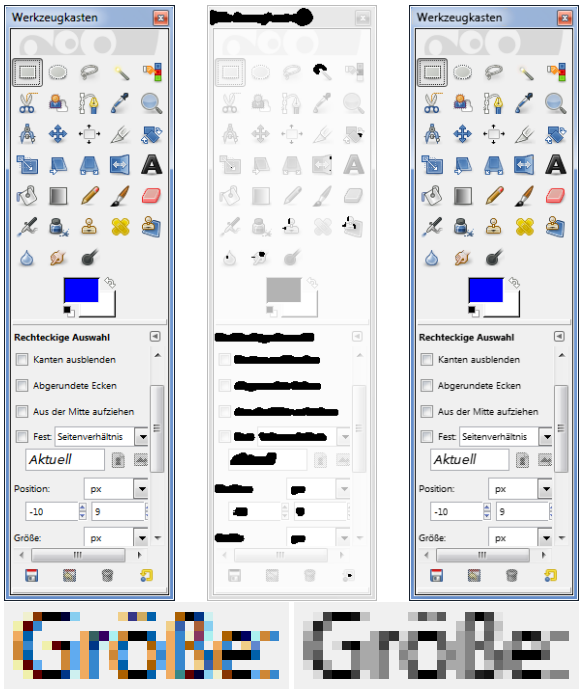

Figure 3: *Processing a screenshot of the Gimp toolbox (left: original; middle: mask; right: processed; bottom: magnified portions).*

References

[BBD*00] BETRISEY C., BLINN J. F., DRESEVIC B., HILL B., HITCHCOCK G., KEELY B., MITCHELL D. P., PLATT J. C., WHITTED T.: Displaced filtering for patterned displays. In *SID Digest*. 2000, pp. 296–299. 1

[GTAE04] GUGERTY L., TYRRELL R. A., ATEN T. R., EDMONDS K. A.: The effects of subpixel addressing on users' performance and preferences during reading-related tasks. *ACM Transactions on Applied Perception 1*, 2 (2004), 81–101. 1

[YJHY07] YE Q., JIAO J., HUANG J., YU H.: Text detection and restoration in natural scene images. *J. Vis. Comun. Image Represent. 18*, 6 (2007), 504–513. 1

© The Eurographics Association 2010.