Efficient, Parallel Scan Enhancement with OpenMP

Aron Gaden (201159)
Friedrich Schiller University Jena
Germany
aron.gaden@uni-jena.de

Bruno Reinhold (202081)
Friedrich Schiller University Jena
Germany
bruno.reinhold@uni-jena.de

ABSTRACT

Scans and document photos commonly have some text or images in the foreground, with the background being of uniform color, mostly white. Often time, the white background is noisy or affected by lighting irregularities. This can make the image harder to read and waste ink if the scan should be printed. While there are some tools with background removal features are available, such as Microsoft Lens, we are not aware of a standalone, open-source solution. In this paper, we propose an efficient, parallelized background removal tool that utilizes contrast enhancement and thresholding. For experimental purposes, the software utilizes no external libraries, except for OpenMP. We achieve a sizeable increase of white portions in the image, while maintaining readabilty of documents.

Author Keywords

noise reduction; background removal; image filter

INTRODUCTION

Scans and pictures from phones generally contain a lot of noise and unnecessary color. The background of a sheet of paper might have a gradient, even though the actual paper background is white, depending on lighting, flash, and other influences. This can reduce the readability of the scan and lead to ink waste when the paper is printed. To solve this problem, we propose a solution using global brightness adjustment and thresholding against a blurred image. Image operations can be computationally expensive. In order to reduce execution time, we aimed to produce cache-coherent code and avoid unnecessary memory access. The implementation also relies on OpenMP[1] for parallelizing convolutions, matrix transposition, and image writing. Apart from OpenMP, only the C++ standard library was used. We opted out of using highly developed and efficient libraries such as OpenCV[2] to get a better practical understanding of implementation details. The scan enhancer currently operates on a simple version of binary *ppm* [2] images for in- and output. While not a particularly efficient file type, it is easily readable and writable, offering optimal properties for a prototype. Our method unsurprisingly produced the best results on clearly separated fore- and backgrounds, but satisfactory results can also be obtained for worse lighting conditions, and for certain circumstances even non-white backgrounds.

[1] https://www.openmp.org/
[2] https://opencv.org/

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

BACKGROUND

The presented algorithm for background removal in images relies heavily on convolution. In the one-dimensional case, convolution can be interpreted as a window moving over the value series, or mathematically as $g(x) = \sum_{s=-a}^{a} w(s) f(x+s)$ where $m = 2a+1$ with m as the filter size, w being an arbitrary filter function, and f as the value series[1]. In our case, we are considering images, which are two-dimensional, so the equation for a filtered image becomes $g(x,y) = \sum_{s=-a}^{a} \sum_{r=-b}^{b} w(s,r) f(x+s, y+r)$. The complexity of two-dimensional convolution amounts to $\mathcal{O}(NMnm)$ with N, M as the image dimensions and n, m as the filter dimensions. The complexity of this operation can be lowered by reducing the two-dimensional convolution to two one-dimensional convolutions, if the kernel is quadratic and of rank 1[1]. In this case, the complexity is reduced to $\mathcal{O}(NM*n+NM*n)$. We can make use of this fact, as we applied Gaussian filters to the image, which are separable. This has additional benefits execution-wise, to which we will come back to in section 2.

METHOD

The final method we implemented consists of multiple reprocessing and filtering layers 1. The resulting pipeline starts by applying a Gaussian filter to the scanned image. The resulting blurred image then acts as the input for a custom contrast enhancing filter provided in the lecture material. The resulting image is already a significant improvement compared to the initial image. However, the background is still very noisy and can be further improved by applying a masking scheme to remove the background entirely, thereby whitening it. The mask is generated by first blurring the result of the contrast filter, followed by a thresholding step. Every pixel above the threshold value becomes part of the white background mask. Thereafter, the mask is applied to the contrast enhanced image of the second processing step, resulting in the enhanced final image.

DEVELOPMENT PROCESS

The development process was handled in two stages: experimenting high-level solutions using Python and Jupyter notebooks, as well as implementing an efficient realization in C++. The filter pipeline we implemented was first prototyped in

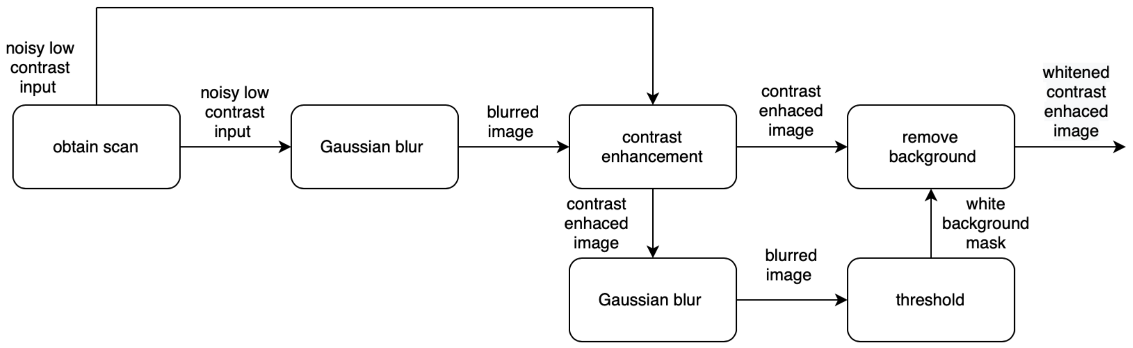

Figure 1. The scan enhancer pipeline.

the Python Programming language. There we experimented with a wide variety of filter and transformation processes. The simplicity of the programming language as well as easy to use software libraries like NumPy and Matplotlib made the prototyping process very efficient. Only after we found a functioning pipeline, we started the implementation of the aforementioned pipeline in C++.
Our first approach was to handle the C++ implementation in an object-oriented manner. We created a class for the ppm images as well as an abstraction of 2D matrices, using vectors. This way, we could represent the RGB channels as three separate parts of a ppm image. After implementing a prototype of the pipeline, we recognized the runtimes to be unacceptable, as figure 1 shows. Note that the naive runtimes do not represent the runtime, but rather just one convolution. We decided to optimize algorithmic bottlenecks first - the biggest runtime factor was the 2D convolution, which is discussed in 1. Separating the filter led us to abandon the inefficient 2D matrix class (with slicing as a big bottleneck) in favor of a regular *std::vector*. This inherently made the program faster, because the memory-locality was improved through the reduction of cross-row accesses. In theoretical considerations, a separated 2D convolution is usually represented by transposing the filter for the second 1D convolution. In programming practice, however, it is preferable to transpose the image instead. This is again related to memory-locality: instead of accessing the elements 1,2, and 3 in the vector, we would be accessing (for example) elements 1, 469, and 937. The former will likely all be on one cache line, while the latter will not properly exploit the cache. Along with the previously discussed complexity reduction for separated convolution, this led to the biggest performance gain. We could further improve the runtime by parallelizing the program. Using OpenMP, it only took minor code adaptations to make the code execute correctly. We found the biggest performance increase for the convolution and transposition. Parallelizing the image writing process led to no noticeable runtime decrease, even for higher-resolution images. The effect of the parallelization heavily differed depending on the system. The macOS with Clang used for the measurements of this paper seemed to only have a small difference in runtime. On an entirely different Linux system with GCC, the parallelization alone halved execution time. Lastly, we replaced the *std::vector* data structure by a 64-Byte aligned vector *aligned_vector* from lecture materials. In architectures using the AVX2 or AVX512 instruction sets, this can reduce overhead when reading or writing RAM.

EXPERIMENTS

Measurements

For measurements, we ran the program on a system with an Apple M1 CPU, running at 4.2 GHz and with 8GB of RAM. The operating system is macOS version 12.2. Table 1 shows the runtime of the scan enhancer. As can be seen, the speedup between a naive implementation and our final result is quite substantial, with convolution, the biggest computation of the program, seeing a speedup of around 6 and around 13 for GCC and Clang, respectively. A comparison to a speed closer to the optimal one could be achieved by creating a reference implementation using a well-optimized library like OpenCV, which we passed on due to time constraints.

Sample Images

The images we chose for the experiments are one with a strong shadow showing on a text document 2. An example of great background removal results is the image in 2. The document retains perfect readability, even in the contoured figure, while removing close to the entire paper background, with some artifacts where the surface of the paper ends, and the table begins.
Figure 3 still shows decent results, and even increases readability due to the contrast enhancement. The large difference in both foreground and background pixels leads to some artifacts in the formerly darker parts of the image. Increasing the contrast less would have led to unreadable text while reducing the artifacts. We decided to value readability highest in our implementation.
The image in figure 4 shows a perhaps interesting edge case: A label with a dark, multicolored background. The scan enhancer removes the dark background somewhat reliably, but the lower part with many color and brightness changes leads to quite noticeable leftover background. If the goal is saving ink for printing this image, the scan enhancer is very successful. Should the background color be kept, the result is not satisfactory. In this case, it may even be necessary to find the different background colors locally - requiring an entirely different approach.

		Runtime Measurements			
Dimensions	*Compiler*	*Naive*[3]	*Efficient*[4]	*Speedup*	*Pipeline*
610x468	GCC	0.078s	0.006s	13	0.453s
610x468	Clang	0.059s	0.009s	6.6	0.624s
3024x4032	GCC	3.41s	0.252s	13.5	2.21s
3024x4032	Clang	2.555s	0.418s	6.1	2.57s

Table 1. Selected runtime measures for the scan enhancer. Averaged over 1000 executions.

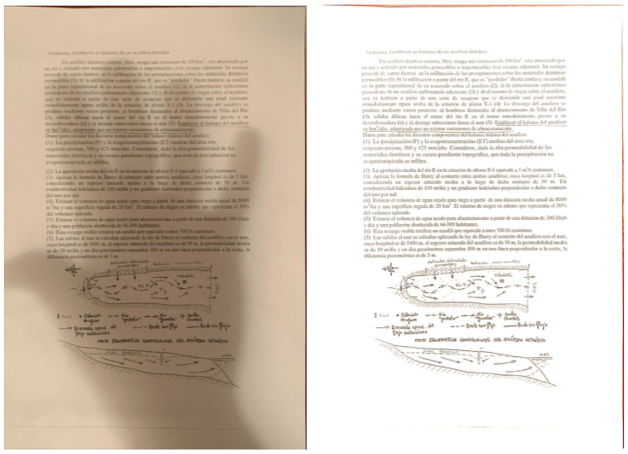

Figure 2. Results on an image with discolored background and shadow.

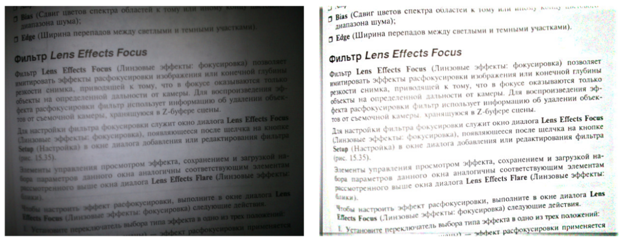

Figure 3. Results on a document with large brightness discrepancies.

CONCLUSION AND FUTURE WORK

The scan enhancer shows promising results for background removal. Good results were attained for somewhat even lighting conditions, even when there are discolorations from light or shadows. For objects with a non-white background, the background will in big parts be replaced by white. Readability of text is still given in those cases. This behavior is especially beneficial, if the goal is to save ink while printing. In case of the background color communicating important information, this may be unwanted. If the brightness in the scan differs greatly, the scan enhancer cannot handle the background perfectly and artifacts may arise. A localized thresholding could improve performance in those cases. Performance-wise, a large speedup could be achieved by optimizing the algorithm,

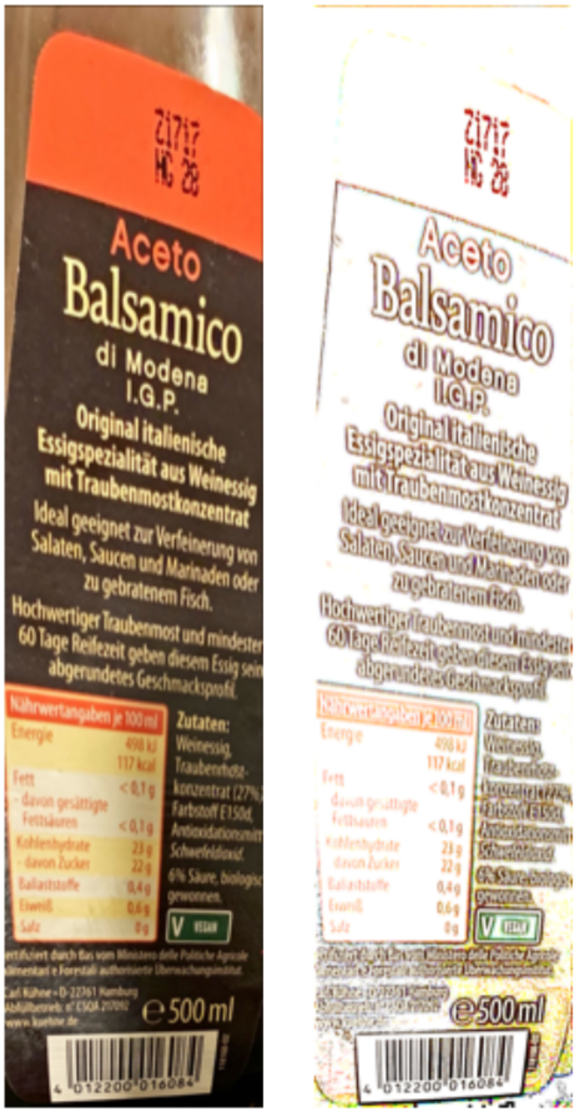
Aceto
Balsamico
di Modena
I.G.P.
Original italienische
Essigspezialität aus Weinessig
mit Traubenmostkonzentrat
Ideal geeignet zur Verfeinerung von
Salaten, Saucen und Marinaden oder
zu gebratenem Fisch.
e500 ml

Figure 4. Results on a label with a dark background.

[3]2D Convolution with a 5x5 Gaussian filter, without parallelization, filter separation and with a Matrix wrapper class for *std::vector<>*
[4]Two 1D separated convolutions with a size 5 Gaussian filter, with parallelization and working with 64 Byte-aligned vectors.

with parallelization leading to the second-biggest improvement. This was also compiler- and system-dependent. The usage of a well-optimized library such as OpenCV could further improve performance and offer more flexibility through a wider array of functionality. To move on from the prototype stage, support for other, more relevant image formats could be added.

REFERENCES

[1] Rafael C. Gonzalez and Richard E. Woods. 2008. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J. `http://www.amazon.com/Digital-Image-Processing-3rd-Edition/dp/013168728X`

[2] Bryan Henderson. 2016. PPM Format Specification. Website. (9 October 2016). Retrieved February 22, 2022 from `http://netpbm.sourceforge.net/doc/ppm.html`.