Improving Optical Character Recognition

AJ Palkovic
Villanova University
alex.palkovic@villanova.edu
United States

ABSTRACT

There is a clear need for optical character recognition in order to provide a fast and accurate method to search both existing images as well as large archives of existing paper documents. However, existing optical character recognition programs suffer from a flawed tradeoff between speed and accuracy, making it less attractive for large quantities of documents. This paper analyzes five different algorithms which operate completely independently of optical character recognition programs, but which have the combined effect of decreasing computational complexity and increasing overall accuracy. Finally, the paper proposes implementing each of these algorithms on the GPU, as well as optical character recognition programs themselves, in order to deliver another massive speed increase.

1. INTRODUCTION

Optical Character Recognition (OCR) is a method to locate and recognize text stored in an image, such as a jpeg or a gif image, and convert the text into a computer recognized form such as ASCII or unicode. OCR converts the pixel representation of a letter into its equivalent character representation. OCR has numerous benefits. Many companies have a large collection of paper forms and documents. Searching these documents by hand may take a long time, and it is only natural to seek to automate this process. One way would be to scan the documents and store them as images on the computer, then perform optical character recognition on the scanned images to extract the textual information into separate text files. Numerous tools for automatic text search through text files already exist. So the main unsolved problem is performing OCR accurately and efficiently. Even online image searches are experimenting with performing OCR on images in their index of websites in order to produce more accurate results. My proposal is to implement accurate OCR enhancements on the Graphics Processing Unit (GPU) as a means to greatly enhancement efficiency without limiting existing accuracy. In the following we discuss a few existing OCR methods. Section 3 presents our research methodologies for implementing the algorithms presented on the GPU.

2. ALGORITHMS TO IMPROVE OCR EFFICIENCY

Despite the benefits of OCR, certain limitations do still exist. In particular, many OCR programs suffer from a tradeoff between speed and accuracy. Some programs are extremely accurate, but are slower as a result. One reason these are slower is because they compensate for a wider variety of documents, such as color or skewed documents. This section presents a number of algorithms which operate independently of OCR programs, but which have the combined effect of decreasing the computational complexity, while increase the accuracy of the programs further.

2.1 BINARIZATION

Modern computers can represent over four billion colors. To represent each color, computers require thirty-two bits then. For color images, this means that every pixel will consume at least four bytes of memory. However, optical character recognition is color independent—a black letter is the exact same as a red letter. Binarization is a method to reduce color images to two colors, black and white. Black and white images only require a single bit per pixel, as opposed to thirty-two for color images. Logically, this greatly reduces the complexity of the image.

2.1.1 THRESHOLD ALGORITHM

One algorithm to perform binarization is the threshold algorithm [1]. This algorithm calculates an arbitrary threshold, T, which is a color. Each pixel's color is compared to the chosen threshold. If the color is above the threshold, then the pixel is converted to a white pixel. If it is below the threshold, the pixel is a black pixel. Although fast and simple, this algorithm has a key flaw. The flaw is the reliance on calculating a single threshold for the entire image. Often the threshold is calculated by averaging the color of every pixel. However, many images may contain very light or dark text which affects the threshold in a negative way. Experimental results showed that low values of the threshold produced letters which appeared to have holes in them, because pixels that should have been black, were chosen to be white. On the

other hand, higher values for the threshold produced blurry characters. One method to fix this flaw is called local binarization [1].

2.1.2 LOCAL BINARIZATION

Rather than calculating a threshold for the entire image at once, local binarization algorithm analyzes each pixel of the image in a small window; as small as five by five pixels. It analyzes each pixel relative to the pixels nearest it in order to convert it into a black or a white pixel. This compensates for variations in text color, as the threshold can be lower for darker text, and higher for lighter text.

2.1.3 RESULTS

In benchmark tests, applying local binarization instead of a global threshold to an image increased the accuracy of OCR, but decreased the running time. The greatest difference was noticed on benchmarks for older documents, as they are the more sensitive to particular global thresholds. For these documents, local binarization improved accuracy up to forty percent, but decreased the running time of OCR by up to thirty-seven percent.

2.2 NOISE REDUCTION

Noise is very common in dirty, wrinkled, or old documents and can alter the result of optical character recognition programs. Noise exists in two forms. ON noise is a black pixel that should be white. In Fig. 1, this would be black pixels on the white background which are not part of a letter. OFF noise is a white pixel that should be black. In Fig. 1, these pixels are the white pixels that are part of letters.

One noise reduction algorithm is morphology [2], and consists of two parts: erosion and dilution. Erosion is a technique to remove ON noise and dilution removes OFF noise. [2]

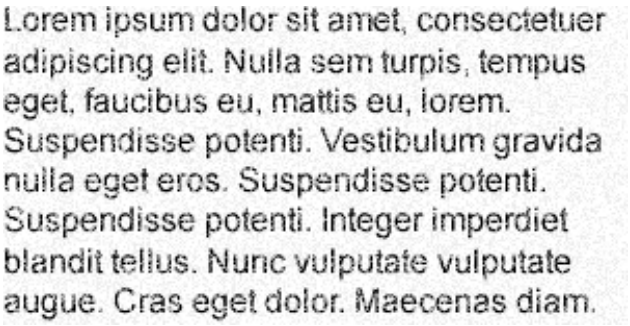

Figure 1

2.3 THINNING

Thinning (see Fig. 2) is an algorithm to further reduce the amount of information in the image to process, thereby reducing the complexity of processing the image [2]. Thinning recognizes that a thick bold letter is the exact same as a letter which is one pixel thick. Thinner letters represent the same information more efficiently.

Thinning is a simple algorithm. Moreover, it is fast and has no flaws. Each row of pixels in the image is scanned left to right. In each row, every sequence of connected black pixels is replaced by a single black pixel in the middle of the sequence. Repeated for the entire image, this technique reduces bold lines to thin, single pixel thick lines.

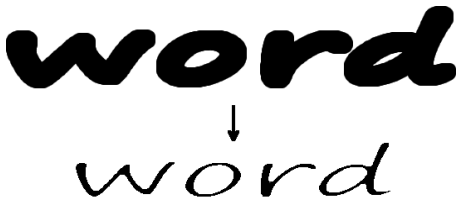

Figure 2

2.3 SKEW DETECTION

When a document is scanned or photographed, some amount of skew inevitably occurs in the scanned document. Even automatic image scanners are unable to perfectly align a document so that it is not tilted one way or the other. This poses a particular problem for the storage and analysis of these documents. It is much simpler to represent a de-skewed document, in which case the information in the document can be stored by the rectangular bounding boxes of document components, such as text. To compensate, one can use either a very complex optical character recognition algorithm to deal with the skew, or detect and correct the skew and then employ a simpler character recognition algorithm.

2.3.1 EXISTING ALGORITHMS

A number of algorithms which detect the angle of skew in a document already exist; however most trade-off between computational efficiency and accuracy. Existing algorithms can be classified into three main categories based on the techniques used for skew detection: projection profile, Hough transformation, and nearest-neighbor clustering [3]. The projection profile technique essentially rotates the document at a variety of angles and then determines the correct angle of the text by analyzing the difference between peaks and troughs in the text. Hough transformations use a voting method to detect defects in objects. Both algorithms are very accurate, especially Hough transformations, however both are unacceptably slow. A third algorithm, based on nearest neighbor clustering, is much faster but succumbs to other limitations; in particular, it is script dependent. A new algorithm has been proposed to correct skew much more quickly by simplifying the document [3].

2.3.2 ENHANCED SKEW DETECTION ALGORITHM

This algorithm is a six step process. First, the image is closed by using a line structuring element (Fig. 3b). This step converts each line of text into a thick black line. However, text often has ascenders and descenders, which are parts of a character that fall above or below the main part of the letter. For example the hook on the bottom of a lowercase 'g', is a descender. The second step removes the ascenders and descenders by opening the image using a small square structuring element (Fig 3c). Next, the entire image is scanned to identify all transitions between white and black pixels, and then the thick black lines created in the previous two steps are replaced by a single pixel thick line. Essentially, this step finds the base of each line of text. Some of the endpoints of the lines contain hooks, as shown in (Fig 3d). The next step trims the lines to eliminate these hooks. After this, the algorithm prunes any short lines remaining. Finally, all of the remaining lines are analyzed for skew and the median value is determined to be the skew angle.

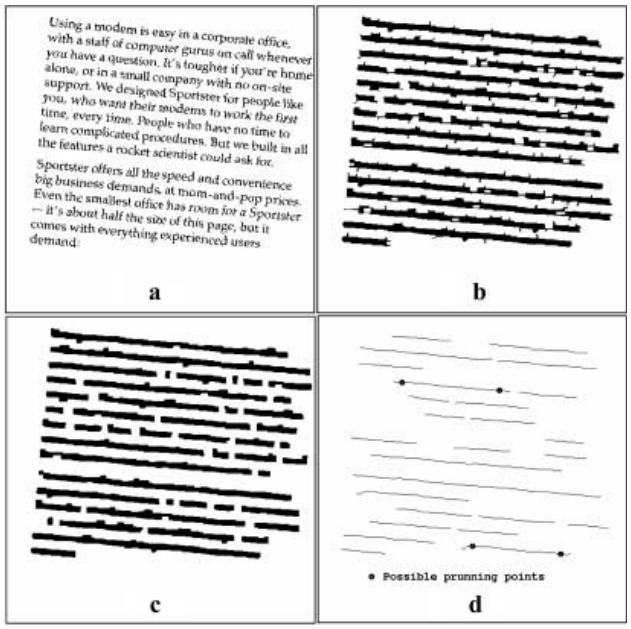

Figure 3 from [3]

2.3.3 RESULTS

Benchmark results [3] showed that the enhanced skew detection algorithm was comparable to the other existing algorithms in terms of accuracy. gThe angle of the skew calculated by the enhanced algorithm was consistently within five percent of the actual angle, better than some of the existing algorithms. For instance, with an actual skew angle of three degrees, the enhanced algorithm calculated the skew of three point eleven degrees.

More importantly, the enhanced skew detection algorithm is much faster than the existing algorithms. Compared to the most accurate of the existing algorithms, Hough Transformations, the enhanced algorithm is over thirty times faster [3].

However, the enhanced algorithm suffered from one major flaw. The algorithm does not properly correct documents skewed more than forty-five degrees. Rather, these will end up skewed ninety, one hundred eighty, or two hundred seventy degrees. At the moment it is unclear how to correct this flaw in the enhanced skew detection algorithm.

2.4 TEXT SEGMENTATION

Segmentation is a method to isolate text in an image. Specifically, it attempts to separate graphics, such as the picture of a tree, from other text contained in an image. It also attempts to separate text from other text, relying on the notion that processing one word or even one letter at a time is faster than processing the entire document at once. Ultimately, segmentation enhances optical character recognition efficiency.

2.4.1 ENHANCED SEGMENTATION ALGORITHM

A fast algorithm for segmenting an image relies on a simple fact about readable text, that characters must have contrast with the background [4]. Gray text on a gray background would clearly be unreadable. As such, this algorithm identifies as potential text, a section of an image that presents a high degree of contrast with the surrounding pixels.

First, the algorithm detects where text is potentially located using existing techniques. It scales the detected text to a 60 point font so that all characters have the same thickness. The algorithm then calculates the normal at each pixel along the border of the character. Then it analyzes the first five pixels on the normal and compares their color to the color of the pixel on the character's border. The algorithm looks for variations between the contrast of the color of the pixels along each of the normals and the color of each of the pixels on the border of the character. The results are separated into clusters and a color reduction method is applied. Finally, a Gaussian mixture model and Bayesian probability are used for each pixel on the border of the character to calculate the probability that the pixel is in a text region.

2.4.2 RESULTS

In experiments, the enhanced text segmentation algorithm proved more accurate than k-means clustering, which is the most common segmentation algorithm. It was consistently five to fifteen percent more accurate in

identifying potential text. It is also expected that the enhanced text segmentation algorithm should be more efficient than k-means clustering, but no time-based benchmark was performed.

3. POST OCR ERROR CORRECTION

Optical character recognition programs do make mistakes in determining which character an image represents. One existing method to correct errors after OCR is to search a dictionary for each recognized word [5]. If a recognized word is not found in a dictionary, it can be replaced by a similar word, which is in the dictionary. However, many words, especially industry-specific jargon, would not be found in a dictionary. As such, this dictionary algorithm results in numerous false alarms.

Test results have shown that over 80% of the errors in optical character recognition result from individual character substitution, insertion, and deletion. This evidence suggests that a better algorithm would analyze the document on a character by character basis. This post OCR algorithm compensates for errors. Rather than representing a word as a combination of just a few letters, this algorithm represents a word as a histogram, giving the probability that the word could be any other word. This means that even if an OCR program mistakes a character, it will be compensated for, because the histogram will show that the incorrect word has a high probability of being the correct word.

This algorithm uses n-grams to expand the possible query terms which could match a document. An n-gram is simply a combination of n letters. A three-gram would be all the combinations of three letters starting with aaa and ending with zzz. For each word, the algorithm calculates the difficultly to transform a word to use each of the n-grams, by calculating the number of characters in a word which would need to be altered and the extent to which they would need to be altered, or *distance*, in order for the n-gram to be in the word. Searching all of these potential n-grams would be time consuming. To compensate, the sequence of all of the n-grams and the distances are stored as a histogram for each word. When searching a document, these histograms are used to match search query terms, instead of the text produced by optical character recognition software. [5]

3. RESEARCH METHODOLOGIES

The goal of this research has been to identify methods to increase the speed of Optical Character Recognition programs without sacrificing accuracy. The following proposed enhancement furthers this goal. GPU is a highly parallel processor. New GPUs from NVIDIA and ATI contain over 200 separate processing cores. [6] Meanwhile, current CPUs only contain up to 4 processing cores per chip. Moreover, GPUs are much more efficient at performing floating point operations. Current GPUs can perform over 1 trillion float point operations per second.[7] Unlike CPUs which perform multiple operations by using loops or function calls, the GPU performs operations by allocating numerous blocks of threads, each of which performs a single small operation. For instance, to perform matrix multiplication, a GPU implementation might allocate one thread to perform each multiplication, whereas a CPU implementation might perform every calculation in a single thread using nested loops.

One way to greatly enhance the speed of OCR is to implement each of the OCR algorithms on the GPU. To date, each of the algorithms analyzed in this research have been implemented for the CPU; however, none have been implemented on the GPU. NVIDIA has produced an API called CUDA [8], which enables developers to implement programs on the GPU using a C-like syntax.

To accomplish this work, the implementations of the algorithms would need to be updated in two ways. First, the implementations need to be transformed from a single threaded, loop-based implementation to one which is highly parallel. The algorithms need to be capable of being broken down into small chunks, each of which can be processed by a separate thread. Second, there are limitations with CUDA which do not exist in C. For instance, the CUDA compiler does not support all of the memory types which are supported by C compilers. Each of the algorithms would need to be revised in light of these limitations.

For instance, the local binarization algorithm calculates the average color of the 25 pixels surrounding each pixel. A GPU implementation might transform this problem, so the resulting color of each pixel was calculated in a separate thread for each pixel. That thread would average the color of the 25 pixels surrounding it and then calculate the resulting color.

The first week of the project would be spent learning the CUDA API more fully, especially its intricacies. Next, I will begin implementing each of the algorithms in three steps. First, I will analyze the existing implementation and plan a new implementation suitable for the highly parallel GPU. Next, I will implementation the new algorithm using CUDA. Finally, I will benchmark the new implementation against the old. The benchmarks will perform the algorithm on a variety of images and compare the average processing time for each. I will proceed using this three step process for each algorithm. Each algorithm will take 1-2 weeks to implement depending on its complexity. In particular, binarization, a simpler algorithm, will take 1 week at most; however, the much more complicated segmentation algorithm will take a longer two weeks.

Finally, I am qualified to engage in this research because of my extensive programming experience. Over the past summer, at an internship with NASA's Jet Propulsion Lab, I studied the CUDA API in order to analyze how it can be used to enhance simulation software, so I am already familiar with this topic. Moreover, much of this research will involve intricate programming problems. This past year, at the International Collegiate Programming Competition, my team tied for 5th in our entire region.

4. OPEN PROBLEMS

Binarization, noise reduction, and thinning are largely solved problems. There is little left to be done to speed up these algorithms. The skew algorithm has two major problems which need to be solved. First, it does not work for documents which are skewed more than forty-five degrees. Although they will be rotated to a multiple of ninety degrees, those documents will not be rotated to zero degrees. Second, the algorithm does not work well if a document contains graphics or images [3]. A number of enhancements are being considered for the n-grams algorithm. First, one can reduce the number of n-grams considered by only using n-grams that occur at least a certain number of times in a dictionary. For instance, zzz and bbz probably do not occur in any word, so should not be considered. Also, a new technique to calculate the difficulty to apply an n-gram to a word is being considered. This technique is iterative rather than the dynamic programming technique currently used [5].

5. CONCLUSION

There is a clear need for optical character recognition in order to provide a fast and accurate method to search both existing images as well as large archives of existing paper documents. However, existing optical character recognition programs suffer from a flawed tradeoff between speed and accuracy, making it less attractive for large quantities of documents.

This paper analyzed six different algorithms to remedy this. The algorithms are able to speed up optical character reocnition as each reduces the complexity of the information to process. For instance, binarization and thinning reduce each letter to the minimum amount of information necessary to still be able to recognize the letter. The algorithms improve accuracy in two ways. First, algorithms like noise reduction and skew correction can reduce the chance of an incorrect match. Second, the n-gram algorithm provides a means to compensate for errors when searching through documents after optical character recognition.

Another large speed increase can be achieved by implementing these algorithms and optical character recognition programs on the GPU. The GPU has considerably more computational power than the CPU; however, it is a much more complex architecture. Each of the algorithms would need to be transformed into a highly parallel solution, but by doing so, optical character recognition can be significantly enhanced.

REFERENCES

[1] Fu Chang, "Retrieving information from document images: problems and solutions," International Journal on Document Analysis and Recognition, Vol. 4, No. 1, August 2001, pp. 46-55, doi: 10.1007/PL00013573.

[2] Rangachar Kasturi, Lawrence O'Gorman and Venu Govindaraju, "Document image analysis: A primer," Sadhana, Vol. 27, No. 1, February 2002, pp. 3-22, doi: 10.1007/BF02703309

[3] A.K. Das and B. Chanda, "A fast algorithm for skew detection of document images using morphology," International Journal on Document Analysis and Recognition, Vol. 4, No. 2, December 2001, pp. 109-114, doi: 10.1007/PL00010902.

[4] Weiqiang Wang, Libo Fu and Wen Gao, "Text Segmentation in Complex Background Based on Color and Scale Information of Character Strokes," Advances in Multimedia Information Processing – PCM 2007, Vol. 4810, 2007, pp. 397-400, doi: 10.1007/978-3-540-77255-2_44.

[5] Y. Fataicha, M. Cheriet, J. Y. Nie and C. Y. Suen, "Retrieving poorly degraded OCR documents," International Journal on Document Analysis and Recognition, Vol. 8, No. 1, April 2006, doi: 10.1007/s10032-005-0147-6.

[6] "Quadro FX 5800", 2008; http://www.nvidia.com/object/product_quadro_fx_5800_us.html

[7] "ATI Radeon™ HD 4800 Series – Overview", 2008; http://ati.amd.com/products/Radeonhd4800/index.html

[8] "What is CUDA", 2008; http://www.nvidia.com/object/cuda_what_is.html