Fast Retinex Image Enhancement Using CUDA

Fast Retinex Image Enhancement Using CUDA

Kyi Myo Zaw[a)], Kunihito YAMAMORI[b)], Masaru AIKAWA[c)], Aye Min Myat[d)]

Abstract

Image enhancement is an important preliminary step in many digital image processing and applications and Retinex algorithm is one of the commonly used algorithm to enhance the image with uneven illumination condition. But the computation of the Retinex is a very complex and time-consuming process. Therefore, we implement the fast Retinex image enhancement algorithm using the CUDA (Compute Unified Device Architecture). Our experiments show that we can gain 46× speed-up for image size 4,096 × 4,096 compared with the OpenCV CPU program.

Keywords: Retinex, NVIDIA CUDA, Image enhancement, Convolution theorem

1. INTRODUCTION

Human Visual System (HVS) can recognize objects and can see the true colour under varying illumination condition, but the imaging devices cannot fully capture the scene like the human eyes. Therefore, the captured images need to be enhanced to get the desired image. Retinex algorithm is commonly used to solve this kind of problems. But the computation of Retinex algorithm is a very intensive and time-consuming process because the processing of the image by Retinex image enhancement has to be performed on each pixel. If this enhancement operation has to be performed sequentially, it will take too much time to complete the enhancement operation.

Nowadays, with the advance in technology, all kinds of image capturing devices can take very high-resolution images and videos. And the computation time of Retinex algorithm is directly proportioned to the size of the image. If the image becomes larger, then the time to perform enhancement operation is increased as well.

The aim of this work is to enhance the speed of Retinex algorithm as fast as possible by using CUDA (Compute Unified Device Architecture) which is a parallel computing platform and application programming interface (API) model created by NVIDIA. Because of Retinex image enhancement can be used as a pre-processing step in many images and videos applications such as face recognition, object detection, vehicle tracking, and other related applications. To work out this, we need to identify which parts of the algorithm could be run in parallel to improve the processing speed and which parts couldn't and how we can improve the processing speed of the algorithm.

a) Master DDP Student, Dept. of Computer Science and Systems Engineering, University of Miyazaki (Master Student, University of Technology (Yatanarpon Cyber City), Pyin Oo Lwin)
b) Professor, Dept. of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki
c) Technical Staff, Faculty of Engineering, University of Miyazaki
d) Associate Professor, Dept. of Computer Engineering, Faculty of Information and Communication Technology, University of Technology (Yatanarpon Cyber City), Pyin Oo Lwin

Our research is constructed as follows: Section 2 expresses the related works and our own methods to approach this problem. Section 3 explains the detailed process of our work and how much speed-up can be got by using our proposed method. Section 4 gives our experimental results and performance analysis. And in Section 5, we conclude our approach with the suggestion on future works and improvements.

2. RELATED WORKS

In this section, there are two topics. Firstly, the use of CUDA in some image processing algorithms and the speedup could be achieved by using CUDA are presented. And then the types of center/surround retinex methods are reviewed one by one.

2.1 Image Processing algorithms using CUDA

CUDA is a parallel computing platform and application programming interface (API) model created by NVIDIA. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing — an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). With the ease of its programmability, CUDA has been adopted to use to accelerate many computationally intensive tasks in image processing and computer vision domains.

A parallel version of Canny edge detector using CUDA was implemented[1)], including all algorithm stages. They achieved 3.8 times acceleration on the CUDA version with the image size of 3,936 × 3,936 resolution compared with an optimized OpenCV version running on a PC. A real-time visual tracker that targets the position and 3D pose of objects in video sequences, specifically faces was implemented by Lozano and Otsuka[2)] and their implementation can achieve 10 times performance improvements as compared with a similar CPU-only tracker.

And according to the parallel image processing based on CUDA[3], as the image size increase, histogram computation can get more than 40 times speedup, removing clouds can get an about 79 times speedup, DCT (Discrete Cosine Transform) can gain around 8 times speedup and edge detection can get more than 200 times speedup.

In the CUDA implementation of McCann99 retienx algorithm by Hyoseok Seo and Ohyoung Kwon[4] the sequential Matlab code takes 25.429 seconds in 1,024 × 1,024 image size, and sequential C codes take 4.1 seconds, and the proposed CUDA implementation takes only 0.760 seconds for the same image. So, CUDA implementation can achieve 5.3 times speedup compared to the CPU implementation written in C.

2.2 Center/surround Retinex

The Retinex algorithm was developed by Edwin Land[5] and is one of the most famous algorithms that attempt to explain the human colour constancy. Colour constancy is one of the most important characteristics of human vision. Because it ensures that the perceived colour to the eyes remains unchanged under varying illumination conditions. The Retinex algorithm is based on the HVS.

There are many Retinex algorithms: path-based Retinex, recursive Retinex, center/surround Retinex and the PDE-based Retinex. Among them, the center/surround Retinex is well suited for parallelization because of the convolution operations, log-domain processing, and normalization process in the algorithm could be run in parallel.

The input image I in the Retinex algorithm is assumed to be formed by a product of the illumination L and the reflection R, i.e., $I = L \cdot R$. The aim of the Retinex algorithm is to estimate the illumination L from the original image I to discard the effect of nonuniform illumination from the image I to improve visual quality of the image I.

The single-scale retinex (SSR) that uses a Gaussian blur operation to compute the center/surround information proposed by Rahman et al.[6]. The extension of SSR is multi-scale retinex (MSR) and that combines dynamic range compression and colour/lightness rendition by using weighted three SSRs with different spatial scales. Finally, the multi-scale retinex with colour restoration (MSRCR) [7, 8] was proposed to restore some colour loss in MSR process using colour restoration factor.

The basic form of single-scale retinex (SSR) is as shown in Eq. 1:

$$R_i(x,y) = \log I_i(x,y) - \log\left[F(x,y) * I_i(x,y)\right], \quad (1)$$

where x and y denote the coordinates of a pixel in image, $R_i(x, y)$ is the Retinex output, $\log I_i(x, y)$ is the image distribution in i-th spectral band, the symbol "$*$" denotes the convolution operator, and $F(x, y)$ is the Gaussian surround function as Eq. 2.

$$F(x,y) = Ke^{-r^2/c^2}, \quad (2)$$

where c is the Gaussian surround space constant, and K is also a constant such that,

$$\iint F(x,y)dxdy = 1. \quad (3)$$

The constant c is used for controlling the scale of $F(x, y)$. A small value of c provides a good dynamic range compression, and a large scale provides better colour rendition.

In order to combine the dynamic range compression and the tonal rendition, SSR is extended to multiscale Retinex (MSR). The result of MSR is a weighted sum of the results of SSR with different scales. The i-th spectral component of MSR output is mathematically expressed by the following equation:

$$R_{MSR_i} = \sum_{n=1}^{N} w_n R_n, \quad (4)$$

where, N : the number of scales,

R_{n_i} : the i-th component of the n-th scale,

w_n : the weight associated with the n-th scale.

R_{n_i} is defined by Eq. 5.

$$R_{n_i}(x,y) = \log I_i(x,y) - \log\left[F_n(x,y) * I_i(x,y)\right], \quad i = 1, \ldots, S, \quad (5)$$

where $F_n(x,y) = Ke^{-r^2/c_n^2}$, c_n is the constant of the n-th scale. From Eq. 4 and Eq. 5, $R_{MSR_i}(x,y)$ can be rewritten as Eq. 6.

$$R_{MSR_i}(x,y) = \sum_{n=1}^{N} w_n \log I_i(x,y) - \log\left[F_n(x,y) * I_i(x,y)\right], \quad i = 1, 2, 3, \ldots, S, \quad (6)$$

where $R_{n_i}(x,y)$ denotes a Retinex output associated with n-th scale for an image, $I_i(x,y)$ and $F_n(x,y)$ denote a surround function. S is the number of spectral bands in the image.

A gain w_n is set to satisfy the condition $\sum_{n=1}^{N} w_n = 1$.

The surround function is given by $F_n(x,y)=K_n e^{-(x^2+y^2)/c_n^2}$, where c_n are the scales that control the extent of the surround (smaller values of c_n lead to narrower surrounds, and larger values of c_n lead to wide surrounds), and the normalization factor is $K_n=\frac{1}{\sum_x\sum_y F_n(x,y)}$.

MSR is good for gray images. But it could be a problem for the colour images because it does not consider the relative intensity of colour bands. This can be seen from the formula of MSR, whose output is the relative reflectances in the spatial domain. Considering the images "out of the gray world", whose average intensity for three colour band are far from equal, the output of MSR for three channels will be more close, which makes it looks more gray. The solution to this problem is to introduce weights for three colour channels depending on the relative intensity of the three channels in the original images. To address the drawback of MSR with regard to colour restoration, we introduced weights for three colour channels depending on the relative intensity of the three channels in the original images. The relative intensity of the three channels is given by Eq. 7,

$$I_i'(x,y)=\frac{I_i(x,y)}{\sum_{n=1}^{S} I_i(x,y)} \tag{7}$$

where I_i is the i-th band of the input image and S is the total number of colour bands. The colour restoration is given as $C_i(x,y)=f[I_i'(x,y)]$.

The best overall colour restoration described in[7] is: $C_i(x,y)=\beta\log[\alpha I_i'(x,y)]$, and then becomes as Eq. 8,

$$C_i(x,y)=\beta\left\{\log[\alpha I_i(x,y)]-\log\left[\sum_{i=1}^{S} I_i(x,y)\right]\right\}, \tag{8}$$

where β is a gain constant, and α is used for controlling the strength of the nonlinearity, S is the number of spectral channels.

The MSRCR is the most computationally extensive among three center/surround retinex algorithms because it requires to compute the three large Gaussian blur convolutions on a large scale to compute the center/surround information for each spectral band.

In our work, we first determine which parts of the algorithm could be implemented in parallel and which parts could not be implemented. We compute the three Gaussian convolutions in the Retinex algorithms in the frequency domain as opposed to the conventional 2D convolution to speed-up the Retinex enhancement process.

3. PROPOSED ALGORITHM USING CUDA

Firstly, we load the image needed to enhance, and classifies whether the image is colour or not, and then apply the CUDA based Retinex algorithm to the image.

We need to classify whether the image is gray scale image or colour image. If the image is gray scale image we only need to enhance the image with MSR. If not, we need to apply the MSRCR algorithm to the image as shown in Figure 1. And Figure 2 shows the image processing operations done in GPU.

3.1 Using Convolution Theorem to Compute the Gaussian Convolution

The process of convolution (a rather messy integral in the spatial domain) has a particularly simple and convenient form in the frequency domain; this is provided by the famous convolution theorem[9].

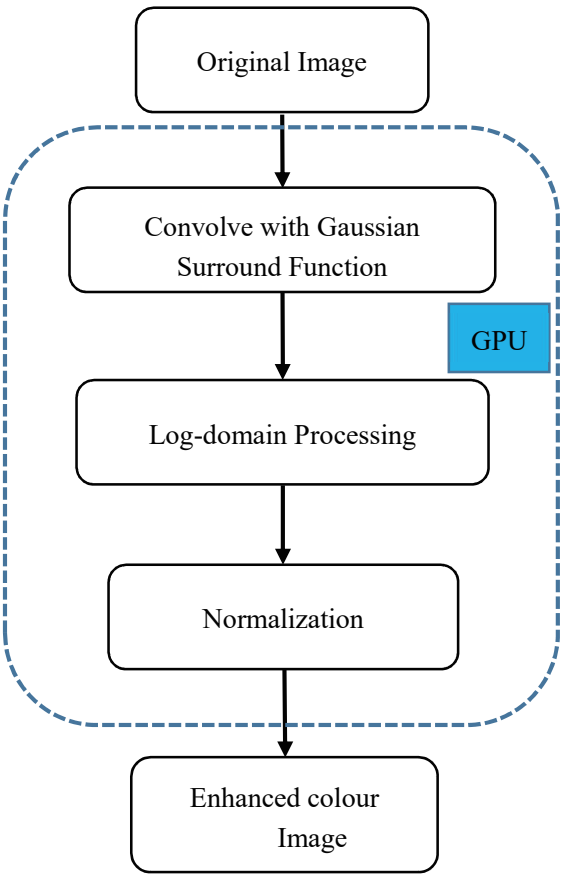

Fig. 1. CUDA operations block diagram.

According to the convolution theorem, the convolution of the two functions by Fourier transform

is equal to the product of the individual transforms. So, the convolution in the SSR becomes as Eq. 9.

$$R_i(x,y) = \log I_i(x,y) - \log\left[F^{-1}\{\hat{F}(v,w).\hat{I}_i(v,w)\}\right], \quad (9)$$

where $\hat{F}(v,w)$ and $\hat{I}_i(v,w)$ are the Fourier transforms of $F(x,y)$ and $I_i(x,y)$ and F^{-1} denotes the inverse Fourier transform.

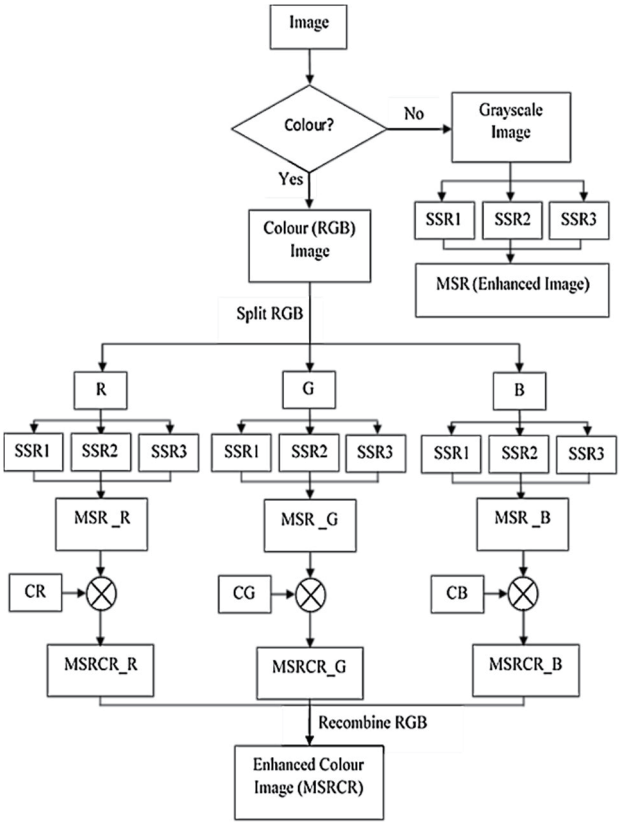

Fig. 2. Overall system block diagram.

Steps for convolution in frequency domain are as follows;

1. Calculate the optimal DFT size for padding.
2. Calculate the Fourier transform of the image with padding.
3. Generate a filter function *F*, the same size as the image.
4. Multiply the transformed image by the filter (pixel-wise multiplication).
5. Inverse Fourier transform to get the convolved image.

Finally, we get the appropriate block size to implemented in the GPU.

3.2 Log-domain processing and normalization

The log-domain processing incudes computing natural logarithm of the original image pixels and the convolved results pixels, addition, subtraction and multiplication. After the computing of the log-domain processing step as shown in Figure 2, we need to normalize the processed pixels from log-domain to display domain those value range is from 0 to 255 for 8-bit colour image because the processed pixels value without normalization have both positive and negative values.

Mooer et al.[10] proposed an automatic treatment of the retinex output prior to display by Eq. 10.

$$R'_{MSRCR_i}(x,y) = \frac{d_{\max}}{r_{\max} - r_{\min}} * (R_{MSRCR_i}(x,y) - r_{\min}), \quad (10)$$

where d_{max} = 255 for 8-bit image, r_{max} and r_{min} are the maximum and the minimum pixel value in the processed image.

4. ANALYSIS AND EXPERIMENTAL RESULTS

The parallel implementations of the convolution, log-domain processing, and normalization are implemented to optimize Retinex image enhancement algorithm. The computational experiments are carried out to compare the performance of the parallel implementation on GPU against the sequential implementation on CPU by using OpenCV.

The computational experiments have been carried out on the Dell Precision Tower 3620 with the following characteristics:

- CPU: Intel (R) Core™ i7-7700 CPU @ 3.6 GHz, 16 GB DDR4 RAM
- GPU: NVIDIA Quadro M4000
  - GPU Memory: 8 GB GDDR5
  - Memory Interface: 256-bit
  - Memory Bandwidth: 192 GB/s
  - NVDIA CUDA Cores: 1,664
  - System Interface: PCI Express 3.0x16
- Window 10 64-bit with Visual Studio 2017 and CUDA toolkit v10.0

Results of the experiments are shown in Table 1 and in Figure 3. We use six resolution of image size for our experiments as shown in Table 1, and measure the total execution time including the data transfer time between CPU and GPU.

We test about 50 images with 10 iterations and take an average to each image sizes to get the execution for both the CPU and GPU. For CPU implementation of the algorithm, we use OpenCV optimized Gaussian blur and other optimized image processing functions. For GPU, we implement it with the combination of OpenCV and CUDA.

As shown in Figure 3, the execution for the smallest image size in this experiment could not achieve speed-up. But as the image size increased, the execution time in CPU is longer and longer as compared to CUDA. This is because the convolution

in Fourier transform could only achieve speed-up for the larger kernel size and image size.

Table 1. The comparison of total execution time between CPU and CUDA (GPU) in milliseconds.

No.	Image Size (M x N)	Total Execution Time (milliseconds)		Speed-up (%)
		CPU	CUDA	
1	256 × 256	463.9	1,501.6	0.31
2	512 × 512	1,786.1	1,550.3	1.15
3	1,024 ×1,024	7,362.7	1,773.0	4.15
4	2,048 × 2,048	45,184.1	2,352.2	19.21
5	3,072 × 3,072	94,698.2	3,242.4	29.21
6	4,096 × 4,096	210,630.0	4,545.5	46.34

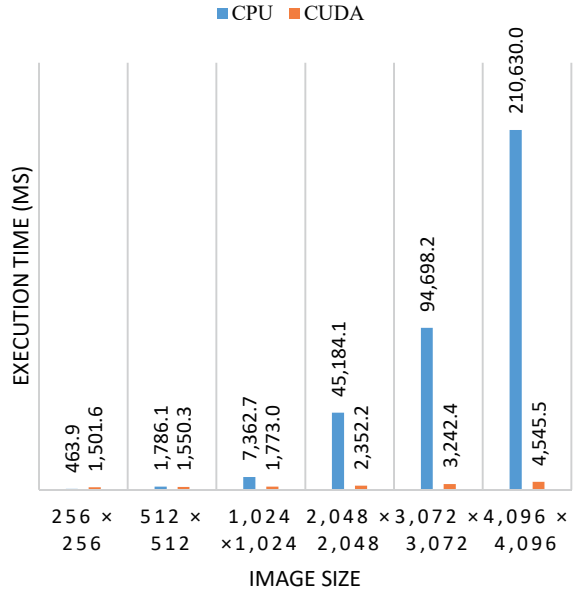

Fig. 3. Total execution time comparison bar chart.

In this experiment, we could not gain speed-up for the 256 × 256 image size compares to other image sizes. Our CUDA implementation for 4,096 × 4,096 resolution image achieved 46.3 times faster than CPU program. So, the proposed algorithm is good for computing large image sizes. Some of the resulted images from the system are shown in Figure 4.

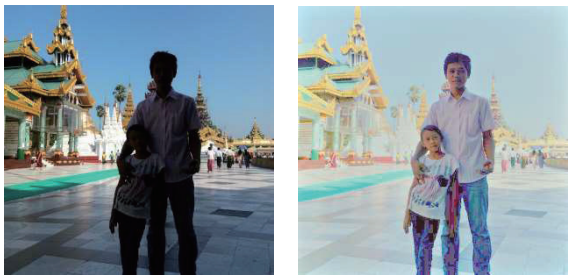

Fig. 4. Experimental results photos: top left and bottom left are the original images and the top right and bottom right are the MSRCR images.

As we can see in Figure 4, the MSRCR images are a little blur compare with the original images, it is an innate flaw of MSRCR and we need some colour balancing techniques to increase the contrast of the image for future processing.

5. CONCLUSION

In this paper, we implemented the fast Retinex algorithm using CUDA and it could achieve more speed-up for the larger image sizes. So, it is well suited for high-speed image enhancement steps for many images and videos applications.

The analysis results could be summarized as follows:

1) We can only get a benefit from the convolution in frequency domain, if the image sizes and kernel sizes are large enough.

2) The enhanced MSRCR images have a blurring effect and that is needed to be solved in the future.

Acknowledgment

Foremost, I would like to express my deepest and sincerest gratitude to my supervisor Prof. Kunihito YAMAMORI for his patience, motivation, empathy, and immense knowledge. His guidance helps me in all time of research and writing of this paper. His dynamism, vision and, sincerity have deeply inspired me. And I would like to thank my friends for helping me to collect the photos needed for experiments.

REFERENCES

1) Y. Luo and R. Duraiswami: Canny edge detection on NVIDIA CUDA, 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008.
2) M. Lozano and K. Otsuka, “Simultaneous and fast 3D tracking of multiple faces in video by GPU-based stream processing,” 2008 IEEE International Conference on Acoustics, Speech and Signal Processing, 2008.

3) Z. Yang, Y. Zhu, and Y. Pu: Parallel image processing based on CUDA, Proceedings of the 2008 International Conference on Computer Science and Software Engineering, 3. 198-201. 10.1109/CSSE.2008.1448.
4) H. Seo and O. Kwon: CUDA implementation of McCann99 retinex algorithm, 5th International Conference on Computer Sciences and Convergence Information Technology, Seoul, 2010, pp. 388-393. doi: 10.1109/ICCIT.2010.5711089
5) E. Land: An alternative technique for the computation of the designator in the retinex theory of colour vision., Proceedings of the National Academy of Sciences, vol. 83, no. 10, pp. 3078–3080, 1986.
6) D. Jobson, Z. Rahman, and G. Woodell: Properties and performance of a center/surround retinex, IEEE Transactions on Image Processing, vol. 6, no. 3, pp. 451–462, 1997.
7) D. Jobson, Z. Rahman, and G. Woodell: A multiscale retinex for bridging the gap between colour images and the human observation of scenes, IEEE Transactions on Image Processing, vol. 6, no. 7, pp. 965–976, 1997.
8 D. J. Jobson: Retinex processing for automatic image enhancement, Journal of Electronic Imaging, vol. 13, no. 1, p. 100, 2004.
9) C. Solomon, T. Breckon, Fundamentals of digital image processing: a practical approach with examples in Matlab. Chichester: Wiley-Blackwell, 2012.
10) A. Moore, J. Allman, and R. Goodman: A real-time neural system for colour constancy, IEEE Transactions on Neural Networks, vol. 2, no. 2, pp. 237–247, 1991.