Single Image Dehazing on Mobile Device Based on GPU Rendering Technology

Yuanyuan Shang

College of Information Engineering, Capital Normal University, 56 Xisanhuanbei Road
Beijing, 100048, P.R. China

Yue Meng

College of Information Engineering, Capital Normal University, 56 Xisanhuanbei Road
Beijing, 100048, P.R. China
Email: syy@bao.ac.cn, mengyue19890119@outlook.com
www.cnu.edu.cn

Abstract

Image dehazing utilizes a very complex algorithm that requires intensive filtering and floating-point arithmetic operations. Consequently, processing speed is the most significant bottleneck in its application in some vision tasks such as mobile platforms. In this paper, we propose an optimized single image parallel processing dehazing algorithm for mobile platforms and implement it on a Windows Phone device based on GPU rendering technology.

Keywords: Fog degradation model, GPU rendering, Image Dehazing, Parallel processing, Windows Phone.

1. Introduction

A variety of approaches to image dehazing has been proposed in the literature. They include titles such as Fast visibility restoration from a single color or gray lever image, by Tarel and Nautiere[1]; Visibility restoration in bad weather from a single image, by Tan[2]; Single image dehazing, by Fattal; and Single image haze removal using dark channel prior, by He[3].

With the continuous improvements in image dehazing, the result of haze removal is getting increasingly better. However, processing speed is the most significant bottleneck to the application of dehazing in some vision tasks, especially on mobile platforms.

Graphic processing unit (GPU) rendering technology is widely used in image processing applications to speed up image processing algorithms. In this paper, we propose an optimized single image parallel processing dehazing algorithm for mobile platforms (based on J.P. Tarel's method), and implement it on a Windows Phone device based on GPU rendering technology. The DirectX programming interface, a powerful tool for accessing GPU resources, can be used on the Windows Phone platform. Thus, we chose it to implement and test the algorithm on the Windows Phone device.

2. Algorithm Overview

The fundamental concept underlying this fast visibility restoration algorithm, which was proposed by Tarel, is the fog degradation model. Koshimider proposed the fog degradation model, and Tarel summarized it as in Equation (1):

$$\mathrm{I}(\mathrm{x},\mathrm{y}) = R(x,y)\left(1-\frac{V(x,y)}{I_s}\right)+V(x,y) \qquad (1)$$

The restored image can thus be obtained by calculating *R(x)*.

V(x, y) in Equation (1) can then be estimated by following the four steps below:

(1) Calculate the minimum value of RGB components *W(x, y)* in the original image:

$$\mathrm{W}(\mathrm{x},\mathrm{y}) = \min_{c\in\{R,G,B\}}(I^C(x,y))$$

where *I (x, y)* is the original input image.

(2) Find the median of *W(x, y):*

$$\mathrm{A}(\mathrm{x},\mathrm{y}) = median_{S_v}(w(x,y))$$

where S_v is the template size of the median filter.

(3) Calculate the difference between *W* and *A*, then calculate the absolute value of the difference:

$$\mathrm{B}(\mathrm{x},\mathrm{y}) = \mathrm{A} - median_{S_v}(\mathrm{abs}(\mathrm{W}-\mathrm{A}))(x,y)$$

where S_v is the template size of the median filter.

(4) Find the maximum value of the minimum:

$$\mathrm{V}(\mathrm{x},\mathrm{y}) = p\max\left(\min\left(B(x,y),W(x,y)\right),0\right)$$

On obtaining a value for *V(x, y)*, the restored image *R(x, y)* can then be calculated using Equation (2), which is derived from Equation (1):[5]

$$R(x,y) = \frac{I(x,y)-V(x,y)}{1-\frac{V(x,y)}{I_s}} \qquad (2)$$

where factor p is used to control the intensity of haze removal, and can be adjusted for different images.

3. Algorithm Parallelization

Some parts of the algorithm cannot be applied to the GPU, in which case the CPU is used. A reasonable division of work between CPU and GPU is the key factor that determines the efficiency of the process.

In this paper, we divide the algorithm into three main parts: white balance, restoration core processing, and tone mapping. Some actions cannot be accomplished in GPU because of its structure. Therefore, we have to transfer these parts of the process to CPU. Accordingly, the white balance algorithm described by Tarel[8] and tone mapping have to be divided into several parts, as depicted in Fig.s 1 and 2, in order to conveniently implement these parts separately in CPU and GPU.

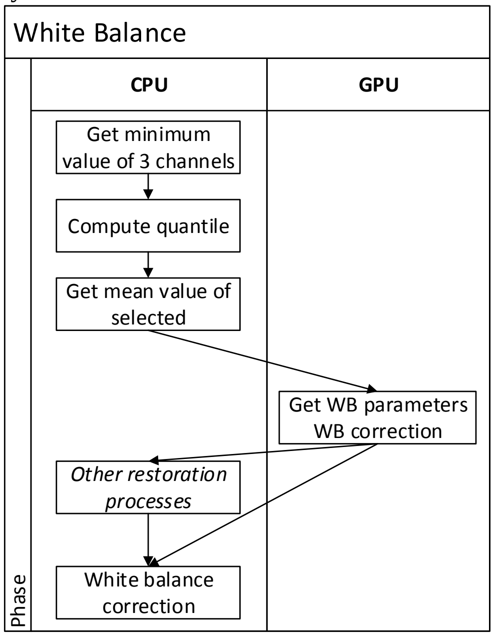
White Balance
Phase
CPU
GPU
Get minimum value of 3 channels
Compute quantile
Get mean value of selected
Get WB parameters WB correction
Other restoration processes
White balance correction

Fig. 1. Division of white balance algorithm between CPU and GPU

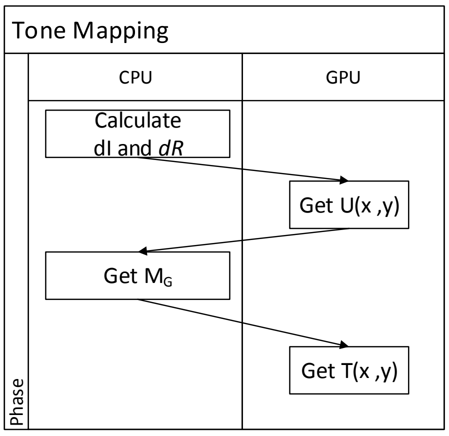

Fig. 2. Division of tone mapping algorithm between CPU and GPU

All processes in the core processing section, which obtains *V(x, y)* in four steps and *R(x, y)* in the final step, can be carried out in GPU; thus, none of this work needs to be transferred to CPU. This is very beneficial for rapid processing. However, because we need to get the process results for neighboring pixels when performing the second median filter to obtain *B(x, y)*, the process has to be split into the two parts: calculating *A* and calculating *R*, as previously discussed. In this process, the final step of the white balance process is combined with the first step of the restoration core processing process, thereby reducing the cost of reassembly.

4. Implementation on Windows Phone

As mentioned before, the GPU processor contains a set of shaders, one of which, pixel shader, has to be programmed in order to implement the image dehazing method. In the DirectX programming interface, a language called high-level shader language (HLSL) is used to program the shaders.[9] A piece of HLSL code corresponds to only a single pixel, not the whole image. All the pixels being processed by the same piece of HLSL program are processed in parallel, rather than sequentially.

Fig. 3. is the workflow chart of the whole algorithm optimized to implement on the GPU, where all textures and parameters have the same name as in the Equation given before. The details of each transfer step were discussed in the previous section.

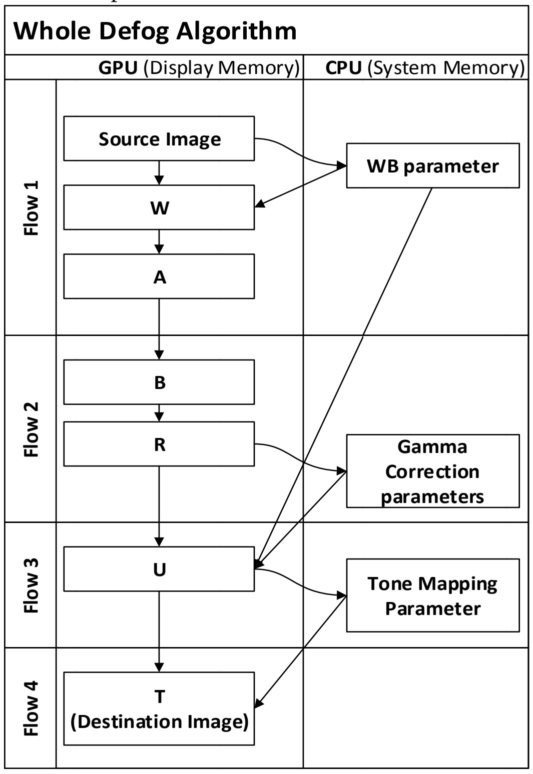

Fig. 3. Workflow of the overall algorithm optimized for G
PU implementation

Note that in the flows across CPU and GPU, curved arrows represent data copies while straight arrows represent parameter transmissions. These are two key techniques in DirectX programming. Data copy refers to the movement of data from display memory to system memory, in order to get access to these data via the usual methods. Parameter transmission is more complex. A class containing the parameters to be transmitted and an ID3D11Buffer to store the parameters in the display memory have to be created. Subsequently, the Context->PSSetConstantBuffer() function has to be used to transmit the parameters to the shader, with the start slot set by this function the same as the register index set in the pixel shader. This is the only interaction between the HLSL and C++ codes.

5. Comparisons and Analyses

Fig. 4 shows the process results. Fig.s 4(a) and 4(d) are the original images, 4(b) and 4(e) are the defog results using GPU, and 4(c) and 4(f) are the defog results using the original algorithm.

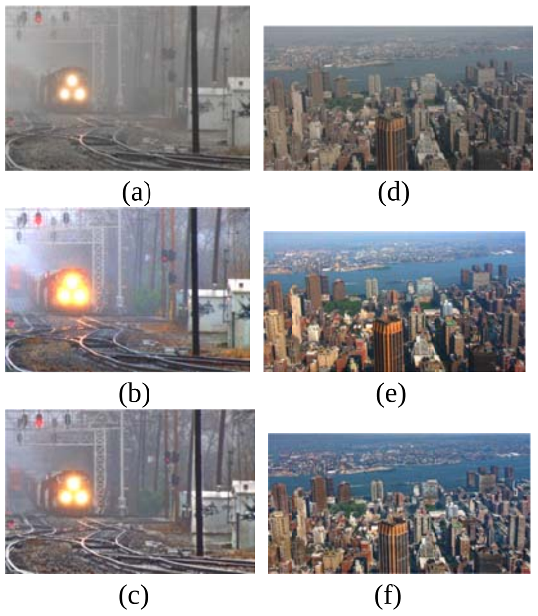

Fig. 4. Experimental results on hazy images

As can be seen in Fig. 4, the process result of GPU processing is very close to that of the original algorithm. In Fig. 2(b), which has more intricate details, the details of the processed image are not as clear as the original algorithm because of changes in the filtering method; however, they are still within the acceptable range. During the implementation, process, we adjusted the intensity of the tone mapping, such that the restored image is more vivid than the image restored by the original algorithm; the color of the image is more realistic.

The focus of our research is improvement of the processing speed. Table 1. shows the processing time of the original algorithm running in CPU and our optimized algorithm running in CPU and GPU. All the tests were conducted on a Windows Phone device with Snapdragon MSM8260A CPU and 1 GB RAM.

Resolution	CPU	CPU+GPU
640 × 480	11.122 s	0.463 s
800 × 600	18.814 s	0.680 s
1024 × 768	33.382 s	1.127 s
1920 × 1080	105.50 s	3.227 s

Table 1. Time cost of CPU only versus CPU+GPU algorithms

In the above chart, it can clearly be seen that there is an improvement in the processing speed when using GPU technology; in the case of the large image, the speed increased as much as 3,000%. This is because we have taken advantage of parallel computing and the floating-point arithmetic of GPU. The most time-consuming part of the original algorithm is bilateral filtering, which becomes a virtual real-time process when running in GPU. Furthermore, other simple operations have also been accelerated in GPU to various degrees. Accelerations of these parts significantly reduced the overall process time of the whole algorithm.

6. Conclusion

With increasing use of image dehazing methods, defogging results are improving significantly. However, the efficiency of the algorithms used in these methods is still very low. GPU technology has developed rapidly in recent years, to the point where general-purpose computing for GPU can be exploited to speed up the image dehazing process. Following parallelization and optimization, we implemented our proposed optimized single image parallel processing dehazing algorithm on a Windows Phone device. Comparisons and analyses show that the efficiency of the algorithm is significantly improved and the defog effect excellent. The algorithm, implemented in GPU, is written as a Windows Phone application and will be published in the Windows Phone app store. In summary, in terms of effectiveness and efficiency, this design has achieved the expected goals.

Acknowledgement

This work is supported by National Natural Science Foundation of China (11178017, 61373090, 61303104, 61203238）, Beijing Natural Science Foundation（4132014）, The Project of Construction of Innovative Teams and Teacher Career Development for Universities and Colleges Under Beijing Municipality, and Beijing Center for Mathematics and Information Interdisciplinary Sciences.

References

1. J. Tarel and N. Hautière, Fast Visibility Restoration from a Single Color or Gray Level Image. ICCV, 2009, pp: 2201–2208
2. R. Tan, Visibility in bad weather from a single image. CVPR, 2008, pp: 1–8.
3. K. He, J. Sun, and X. Tang, Single image haze removal using dark channel prior. CVPR, 2009, pp: 1956–1963.
4. J. Tarel and N. Hautière, Fast Visibility Restoration from a Single Color or Gray Level Image. ICCV, 2009, pp: 2201–2208.
5. Yue Meng and Yuanyuan Shang, Design of Real-time Haze Image Restoration System Based on FPGA Technology. Information Technology Journal, 2013, pp: 7481–7488.
6. J. Tarel and N. Hautière, Fast Visibility Restoration from a Single Color or Gray Level Image. ICCV, 2009, pp: 2201–2208.
7. P. Shirley, J. Ferwerda, E. Reinhard, and M. Stark. Photographic tone reproduction for digital images. In ACM SIGGRAPH’ 02, 2002, pp:267–276.
8. J. Tarel and N. Hautière, Fast Visibility Restoration from a Single Color or Gray Level Image. ICCV, 2009, pp: 2201–2208.
9. HLSL and Pixel Shader for XAML Developers, Walt Ritscher. O’reilly Media, 1st edition July 2012