Distortion correction algorithm for UAV remote sensing image based on CUDA

ZHANG Wenhao[1,2], LI Yingcheng[1,2], LI Delong[1,3], TENG Changsheng[1] LIU Jin[1,3]

[1]China TopRS Technology Co., Ltd, Beijing 100039, China
[2]Chinese Academy of Surveying and Mapping, Beijing 100039, China
[3]Beijing Institute of Civil Engineering and Architecture, Beijing 100044, China

E-mail: wenhao121@163.com

Abstract. In China, natural disasters are characterized by wide distribution, severe destruction and high impact range, and they cause significant property damage and casualties every year. Following a disaster, timely and accurate acquisition of geospatial information can provide an important basis for disaster assessment, emergency relief, and reconstruction. In recent years, Unmanned Aerial Vehicle (UAV) remote sensing systems have played an important role in major natural disasters, with UAVs becoming an important technique of obtaining disaster information. UAV is equipped with a non-metric digital camera with lens distortion, resulting in larger geometric deformation for acquired images, and affecting the accuracy of subsequent processing. The slow speed of the traditional CPU-based distortion correction algorithm cannot meet the requirements of disaster emergencies. Therefore, we propose a Compute Unified Device Architecture (CUDA)-based image distortion correction algorithm for UAV remote sensing, which takes advantage of the powerful parallel processing capability of the GPU, greatly improving the efficiency of distortion correction. Our experiments show that, compared with traditional CPU algorithms and regardless of image loading and saving times, the maximum acceleration ratio using our proposed algorithm reaches 58 times that using the traditional algorithm. Thus, data processing time can be reduced by one to two hours, thereby considerably improving disaster emergency response capability.

1. Introduction

In recent years, Unmanned Aerial Vehicle (UAV)-based, low-altitude remote sensing systems have played an important role in major natural disaster emergencies, and they have become an important technique of obtaining disaster information. Sensors in low-altitude UAV systems are typically non-metric digital cameras having optical distortion, which is an important error that could affect the quality of pixel coordinates. Optical distortion also affects the precision of image matching, aerial triangulation orthorectification and mosaicing, etc. Distortion correction can eliminate the influence of lens distortion, and thereby provide accurate initial data for subsequent processing. Distortion correction, therefore, has become an indispensable foundation work in image processing in UAV remote sensing systems.

During distortion correction, it is necessary to process large amounts of data. For any district, the number of images that obtained by low-altitude UAV remote sensing systems with high resolution, small format and large overlap could run in thousands[1][2]. The traditional correction method is based on a pixel-by-pixel calculation of low computational efficiency. Using the traditional method, it is difficult to meet the demand for fast processing, and processing speed has become one of the bottlenecks that need to be resolved in order to improve the efficiency of image processing in low-altitude remote sensing image systems. In recent years, the progress of the Graphics Processing Unit (GPU) has greatly exceeded the Moore's law. When compared with CPU algorithms in parallel computing, distributed computing, and floating-point operations, GPU's computing power has proven to be dozens of times or a hundred times faster[3]. At the same time, GPU's highly parallel computing power for general computing has gradually become a hot topic. Researchers have implemented Compute Unified Device Architecture (CUDA) for image and video codec and HD video conversion, and they are carrying out research in mathematical calculations, image processing, computational biology and chemistry, fluid dynamics simulation, seismic analysis, ray tracing, and so on[4].

This paper shows the results of studies performed with a parallel processing algorithm for distortion correction based on GPU that uses CUDA to implement the algorithm. The experiments show that the proposed CUDA-based distortion correction algorithm is correct. When not taking into account image loading and saving times, speedup was 58 times than it would have been with the traditional method. The study thus shows that the CUDA-based distortion correction algorithm considerably improves the efficiency of distortion correction, and fully meets the requirements of disaster emergency data processing.

2. Low-altitude UAV remote sensing image distortion correction model

A UAV remote sensing system with a non-metric digital camera distortion correction model is given as follows.

$$\Delta x = (x - x_0)(k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2(x - x_0)^2) + 2p_2(x - x_0)(y - y_0) + \alpha(x - x_0) + \beta(y - y_0)$$
$$\Delta y = (y - y_0)(k_1 r^2 + k_2 r^4 + k_3 r^6) + p_2(r^2 + 2(y - y_0)^2) + 2p_1(x - x_0)(y - y_0)$$

where $\varDelta x, \Delta y$ –are the correction of the image point coordinates; x, y –are the coordinates of the image point; x_0, y_0 – are the principal point coordinates; k_1, k_2, k_3 – is the radial distortion; p_1, p_2 – is the decentering distortion; α, β – is the distortion in array; and $r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$

3. CUDA architecture and programming model

CUDA is a system proposed by NVIDIA that is comprised of software and hardware, and that treats the GPU as a parallel computing device. In a CUDA programming environment, the CPU, as the host, is responsible for the logical transaction processing and serial computations, such as thread creation, the application of graphics memory and data storage, etc. GPU, as the device, is responsible for the implementation of highly threaded parallel processing tasks, memory management and so on. GPU general computing, based on the CUDA programming model, is divided into three steps:

1) Allocate space in the graphics memory, and copy the data from the memory to the graphics card;
2) Start kernel function, and process data on the GPU;
3) Copy the result back to the processing memory.

The GPU threads are the minimum execution units, multiple threads (typically set to a multiple of 16) form a thread block, and multiple blocks can be composed of thread grid, where grid is the execution unit. Therefore, a large number of threads achieve parallel computing. CUDA uses a Single Instruction Multiple Thread (SIMT) execution model. In the same block, all threads are executed in parallel with a kernel function. At any time, each CUDA kernel executes the same instruction, but operates on different data.

4. Distortion correction algorithm based on CUDA

4.1. Algorithm analysis

In this study, we used the indirect correction algorithm. Using known camera distortion correction parameters and distortion correction model, we calculated the corrected image for each pixel location in the original image. Then we calculated the pixel value from the original image using an interpolation algorithm, and we assigned the value to the corresponding position of the corrected image. Finally, the image was saved to complete the distortion correction process.

From the analysis described above, one can see that there are two parts related to large amounts of data procession: one is the calculation of error correction values, and the other is the interpolation of pixel values. These can be carried out on the GPU to improve computational efficiency.

4.2. Algorithm design

According to the analysis described in section 4.1, one can see the calculation of the correction value and interpolation are computationally intensive and time-consuming tasks. Therefore, these two tasks were calculated at the GPU side. The image was read and stored in the CPU end. In addition, the GPU's constant memory store distortion parameters and texture memory store image. The flow of the algorithm is shown in Figure 1.

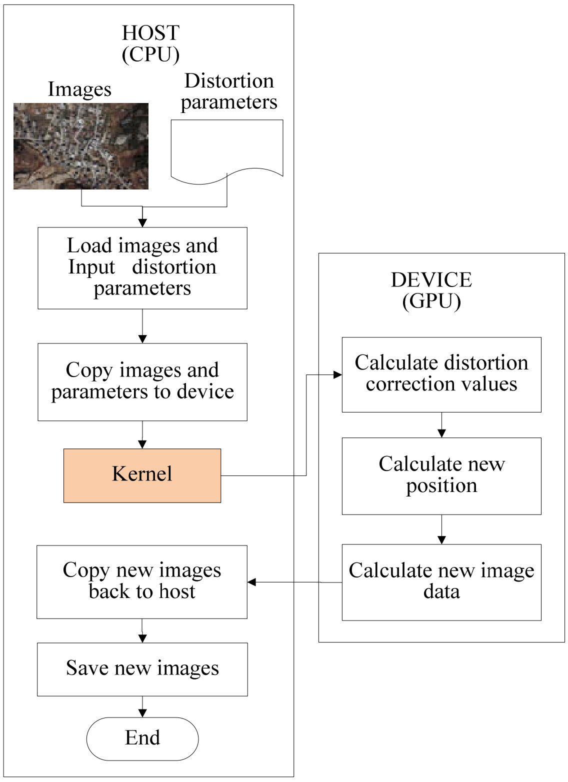

HOST
(CPU)
Images
Distortion parameters
Load images and Input distortion parameters
Copy images and parameters to device
Kernel
Copy new images back to host
Save new images
End
DEVICE
(GPU)
Calculate distortion correction values
Calculate new position
Calculate new image data

Figure 1. Flow chart for the CUDA-based distortion correction algorithm

5. Experimental testing and analysis

5.1. Experimental computing platform

Hardware platform: 4-core Intel (R) Xeon (R) CPU W3550@3.07GHz, system memory up to 4.0GB, NVIDIA Quadro 4000, memory 2GB.

Software platform: Microsoft Windows XP (32-bit), Visual Studio 2008, CUDA Computing Toolkit 4.0.

5.2. Precision testing and analysis

Experiment 1: Experimental data were composed of outdoor calibration field images of the Chinese Academy of Surveying and Mapping. A total of 52 images were obtained using a Canon EOS 5D Mark II camera. The camera calibration software, named Australis, was used to obtain the camera calibration parameters. Using the CUDA-based distortion correction algorithm and camera calibration parameters, we corrected the calibration of the experimental images. Further, using the Australis software, we obtained the calibration parameters of the corrected images. The results are summarized in Table 1.

Table 1. Comparison of calibration parameters before and after correction

Parameter	Before correction	After correction	Comments
x_0	-0.0146	-0.0009	Principal point (mm)
y_0	-0.1387	0.0006	
k_1	7.11402e-005	-7.67683e-008	Radial distortion
k_2	-6.00077e-008	2.14936e-010	
k_3	-1.88873e-011	-3.22840e-013	
p_1	-4.5859e-006	5.1164e-007	Decentering distortion
p_2	9.9744e-006	-2.1298e-007	
α	1.4600e-004	1.5589e-006	Distortion in array
β	3.9291e-005	-1.2953e-006	

As can be seen from Table 1, the distortion parameters of the corrected images are greatly reduced, thus fulfilling the purpose of distortion correction.

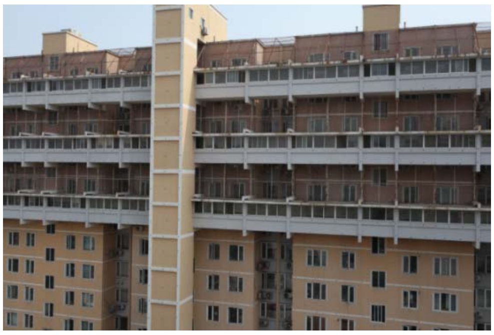
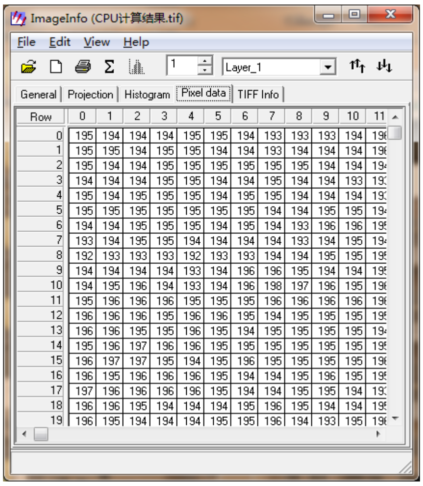


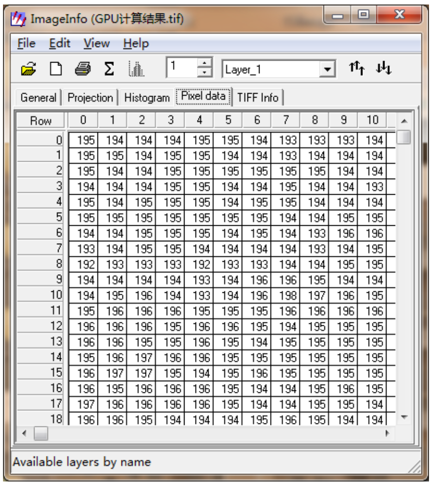
Figure 2. Original image



ImageInfo (CPU计算结果.tif)

Row	0	1	2	3	4	5	6	7	8	9	10	11
0	195	194	194	194	195	195	194	193	193	193	194	196
1	195	195	194	195	195	194	194	193	194	194	194	196
2	195	194	194	195	195	195	195	195	195	194	194	194
3	194	194	194	195	195	194	194	195	194	194	193	193
4	195	194	195	195	195	194	195	195	194	194	194	193
5	195	195	195	195	195	195	195	194	194	195	195	194
6	194	194	195	195	195	194	195	194	193	196	196	195
7	193	194	195	195	194	194	194	194	193	194	195	194
8	192	193	193	193	192	193	193	194	194	195	195	195
9	194	194	194	194	193	194	196	196	195	194	194	195
10	194	195	196	194	193	194	196	198	197	196	195	196
11	195	196	196	196	195	195	195	196	196	196	196	196
12	196	196	196	195	196	196	195	194	195	195	195	195
13	196	196	195	195	196	195	194	195	195	195	195	194
14	195	196	197	196	196	195	195	195	195	195	195	195
15	196	197	197	195	194	195	196	195	195	195	195	196
16	196	195	196	196	196	195	194	194	195	196	195	195
17	197	196	196	196	196	195	194	194	195	195	194	193
18	196	196	195	194	194	194	195	196	195	194	194	195
19	196	195	194	194	194	195	195	196	194	193	195	196

Figure 3. Result of CPU



ImageInfo (GPU计算结果.tif)

Row	0	1	2	3	4	5	6	7	8	9	10
0	195	194	194	194	195	195	194	193	193	193	194
1	195	195	194	195	195	194	194	193	194	194	194
2	195	194	194	195	195	195	195	195	195	194	194
3	194	194	194	195	195	194	194	195	194	194	193
4	195	194	195	195	195	194	195	195	194	194	194
5	195	195	195	195	195	195	195	194	194	195	195
6	194	194	195	195	195	194	195	194	193	196	196
7	193	194	195	195	194	194	194	194	193	194	195
8	192	193	193	193	192	193	193	194	194	195	195
9	194	194	194	194	193	194	196	196	195	194	194
10	194	195	196	194	193	194	196	198	197	196	195
11	195	196	196	196	195	195	195	196	196	196	196
12	196	196	196	195	196	196	195	194	195	195	195
13	196	196	195	195	196	195	194	195	195	195	195
14	195	196	197	196	196	195	195	195	195	195	195
15	196	197	197	195	194	195	196	195	195	195	195
16	196	195	196	196	196	195	194	194	195	196	195
17	197	196	196	196	196	195	194	194	195	195	194
18	196	196	195	194	194	194	195	196	195	194	194

Available layers by name

Figure 4. Result of GPU

Experiment 2: The purpose of the second experiment was to test the differences in the results of the CPU and GPU using the correction algorithm based on CPU and CUDA, respectively, to correct the data in Experiment 1. The original image is shown in Figure 2. The results are shown in Figures 3 and 4.

As can be seen in Figures 3 and 4, the CPU and GPU calculation results are entirely consistent, with the same accuracy.

5.3. Efficiency testing and analysis

The experimental data were obtained by a low-altitude UAV remote sensing system. The image size was 5616*3744 pixels, and the image format was JPG. To conduct comparative experiments on multiple sets of data, our first step was to crop the original images. The cropped image sizes were 8520*7120, 7120*5010, 6550*4080, 5616*3744, 4080*3720, 3840*2560, 2880*1920, 1920*1280, 1280*720, 720*480, and 380*240, for a total of 11 sets of experimental data. Each set consisted of 50 images. CPU and GPU were used to correct these 11 sets of experimental data; we recorded processing time, but not take into account the time required to load and save the images. The processing times are listed in Table 2. Figure 5 shows the GPU and CPU speedup.

Table 2. Processing time of CPU and GPU for different image sizes

Image size(pixel)	CPU time(ms/image)	GPU time (ms/image)	Speed up
380*240	18	1	18.00
720*480	68	2	34.00
1280*720	206	5	41.20
1920*1280	601	14	42.93
2880*1920	1173	23	51.00
3840*2560	2610	50	52.20
4080*3720	4437	79	56.16
5616*3744	6162	109	56.53
6550*4080	8129	143	56.85
7120*5010	10722	185	57.96
8520*7120	16895	291	58.06

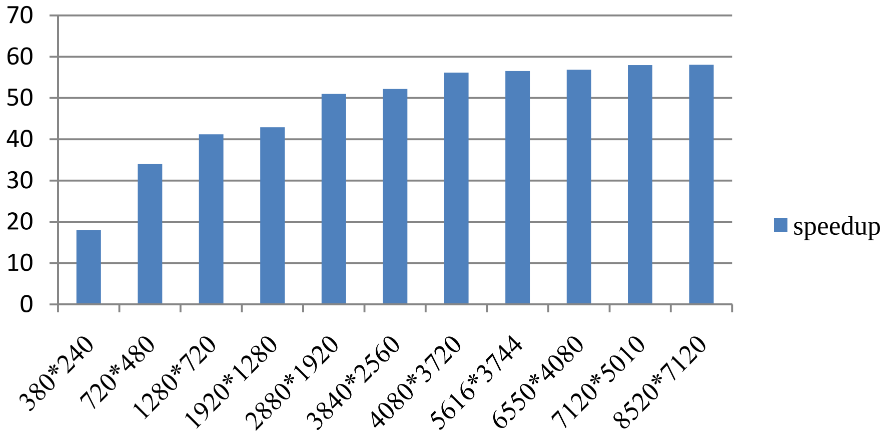


Figure 5. GPU and CPU speedup

As can be seen from Table 2 and Figure 5, with the image size increases, the GPU acceleration effect is very obvious. The speedup reached 56 times that of the traditional speed, when the image size increased to 4080*3720 pixels. With continued increase in image size, owing to bandwidth limitations, the data transfer between the CPU and GPU increased, and speedups slowed down.

6. Conclusion

Distortion correction is time-consuming and affects the processing efficiency of low-altitude UAV remote sensing systems, thereby restricting their rapid response capability. We proposed an image distortion correction algorithm for CUDA-based low altitude UAV remote sensing. Tests show that the algorithm is correct and fully meets the accuracy requirements of distortion correction. Without considering image loading and saving times, the maximum processing speed can reach 58 times the speed of the traditional CPU algorithm, greatly improving the efficiency of distortion correction. the difference in speed is of great significance for improving the efficiency of image processing in low-altitude UAV remote sensing, which in turn can help exploit to the advantages of UAV remote sensing systems, as well as improve the response capacity to disasters and emergencies.

Acknowledgments

This paper was financially supported by Program “Key technologies and series equipment development of emergency mapping aerial remote sensing data obtain” (2012BAK15B03) and Program “Real-time solver technology of the air sensor position and attitude on the plane” (2011BAH12B02).

References

[1] Cui Hongxia 2005 Research of non-metric digital camera distortion Calibration, *Science of Surveying and Mapping*,**30**, 47-56.

[2] Lin Zongjian, Su Guozhong and Zhi xiaodong 2006 UAV borne low-altitude Photogrammetry system with composite double camera, *Geospatial information*, **4**, 1-3.

[3] Manavski S A 2007 CUDA compatible GPU as an efficient hardware accelerate for AES cryptography, *IEEE International Conference on Signal Processing and Communications*, 65-68.

[4] Sudipta N Sinha, Jan-Michael Frahm and Marc Pollefeys 2006 GPU-Based Video Feature Tracking and Matching,*Department of Computer Science*,06-012.