2013 Seventh International Conference on Image and Graphics

Accelerating Harris Algorithm with GPU for Corner Detection

Shuhua Luo, Jun Zhang
School of Information Science and Engineering, Central South University
Hunan Engineering Laboratory for Advanced Control and Intelligent Automation
Changsha, China
luoshuhua10@163.com, junzhang@mail.csu.edu.cn

***Abstract*—Harris algorithm is a corner detection method based on gray scale, which detects corner points by calculating the gradient changes of pixels in horizontal and vertical direction. Due to the usage of Gaussian filtering, Harris algorithm performs well on aspect of robustness, accuracy and stability. A number of improved Harris algorithms have been proposed to enhance the accuracy and robustness during recent decades. However, the execution speed is limited by a large quantity of calculations especially for Gaussian smoothing, which has remained to be a bottleneck in real-time vision processing. Graphics processing unit (GPU) has special parallel computing resources in hardware. Single Instruction and Multiple Data (SIMD) architecture of shader processor array is suitable and fast for mass data calculations in parallel. Therefore, it provides us with an alternative implementation on GPU which can solve such a bottleneck problem. The paper proposes an implementation of Harris corner detection algorithm on GPU by using open graphics library (OpenGL) and graphics library shading language (GLSL) for portability. The experiment results show that the full execution performance of the implementation on GPU is over 73 times speedup of that completely on CPU at most, and can meet the requirements of real time, accuracy and robustness.**

Keywords-Harris algorithm; corner detection; GPU ; OpenGL

I. INTRODUCTION

Corner is an important local feature point of images. Corner detection is helpful to find out the target, and is usually used as a preparation for the next step of target matching and tracking. It is widely applied in computer vision processing, for example, motion object tracking, virtual scene reconstruction, and image registration and so on. At present, the methods of feature point detection are classified as the methods based on the edge feature and the methods based on gray scale. Kitchen-Rosenfeld [1], CSS [2], Freeman chain code [3] and Wang-Brady [4] are the first class, which have difficulty in extracting feature and calculating a large scale of data. Nevertheless, the methods based on gray scale are simple and direct, more suitable for rapid detection. Harris corner detection algorithm [5] belongs to that, which calculates the gray level gradient of pixels to detect corner point. The main process is to take initial smoothing on the gray scale image which is transformed by source color image, then calculate the interest values of pixels after utilizing Gaussian filtering window to smooth gray scale image. If the interest value exceeds certain threshold value, then take the point as a corner point and select a local extreme point within an adjacent area.

With the rapid development of computer vision technology, precision, robustness and real time are increasingly emphasized. A number of improved algorithms were suggested to meet such requirements. Guo Chenguang etc. proposed an improved Harris algorithm in which the corner response function of MIC algorithm was integrated to reduce the calculation amount of Gaussian smoothing link, and a comparison function was added to exclude some disturbance points nearby real corners [6]. Zhao Wanjin etc. proposed a novel Harris multi-scale corner detection algorithm based on wavelet, in which the corners was detected in different scales [7]. It could overcome the drawback of missing significant corners and detect false corners caused by noise. The ideology of image blocks and neighboring point eliminating method were used to improve the robustness of Harris algorithm [8]. It made the corners well-proportioned distributing, and effectively avoided clustering too many corners together. However, those optimized algorithms described above only consider how to improve the accuracy and robustness, and less attention is paid to how to meet the real time requirement. What's more, in order to obtain high accuracy, more and more data need be processed. All the above algorithms are executed on CPU which has the characteristic of sequential processing, but the final result of performance improvement is still limited. Consequently, in order to achieve higher performance, those algorithms need a high-frequency multi-core processor or combine a number of multi-core processors. However, the cost is too high to purchase that kind of processors plus its infrastructure. Fortunately, GPU has special parallel computing resources in hardware, which can be utilized to take normal graphics rendering and accelerate the non-graphics applications especially the parallel matrix calculations. A practice of accelerated mean shift algorithm with GPU has shown that the processing speed of the algorithm implemented on GPU can be high speed up of that completely on CPU [9]. Though Harris algorithm implementation with Computer Unified Device Architecture (CUDA) has also proved GPU's accelerating effect [10], it can't meet the requirement of portability because CUDA has some limits to graphics card and GPU chips. Since OpenGL and GLSL have a platform-independent generality, they

978-0-7695-5050-3/13 $26.00 © 2013 IEEE
DOI 10.1109/ICIG.2013.36

provide us with an alternative implementation on GPU to accelerate Harris corner detection algorithm for portability.

The purpose of this paper is to propose an implementation of Harris corner detection algorithm at a high speed on GPU by using OpenGL and GLSL for portability. The remainder of this paper begins with an introduction of Harris corner detection algorithm in Section 2. Section 3 describes the implementation on GPU with OpenGL and GLSL in detail. The following is experiments and evaluation in Section 4. Discussions about performance optimization of the implementation on GPU are presented in Section 5 and Section 6 concludes the paper.

II. HARRIS ALGORITHM

A. Algorithm Description

Harris algorithm is a corner detection method based on gray level, which detects corner points by calculating the gradient changes of pixels in horizontal and vertical direction. The principle of corner detection is to move the image processing window w (generally rectangular area) to any small shift direction (x, y) and see if the gray scale gradient changes of two directions are both large. If both of them are very small, it's the flat in the image; if one is much larger than the other, it's the edge; if they are both large, it's the corner. The quantity change equation can be defined as following:

$$\begin{aligned} E_{x,y} &= \sum w_{u,v}\left[I_{x+u,y+v}-I_{u,v}\right]^2 \\ &= \sum_{u,v} w_{u,v}\left[xX+yY+O\left(x^2+y^2\right)\right]^2 \\ &= Ax^2+2Cxy+By^2 \\ &= (x,y)\,M\,(x,y)^T \end{aligned} \tag{1}$$

The main calculations of Harris algorithm are generally divided into five steps as following:

Step 1: transform the color image to gray scale image by using (2).

$$\text{grayscale} = R*0.299 + G*0.587 + B*0.114 \tag{2}$$

Step 2: calculate the first-order gray scale gradient in horizontal and vertical direction and take initial smoothing with (3) and (4).

$$X = \frac{\partial I}{\partial x} = I \otimes (-1,0,1) \tag{3}$$

$$Y = \frac{\partial I}{\partial y} = I \otimes (-1,0,1) \tag{4}$$

Step 3: filter the image with the Gaussian window in order to improve the ability to resist noise according to (5), (6) and (7).

$$w_{u,v} = \exp\left[-\frac{u^2+v^2}{2\sigma^2}\right] \tag{5}$$

$$A = X^2 \otimes w, B = Y^2 \otimes w, C = (XY) \otimes w \tag{6}$$

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{7}$$

Step 4: calculate the interest value of pixels using (8).

$$R = A*B - C^2 - k*(A+B)^2 \tag{8}$$

k is experience value, and usually set from 0.04 to 0.06.

Step 5: select local extreme point within an adjacent area as the real corner point.

B. Algorithm Analysis

Harris corner detection algorithm has a simple operation because it only uses the first-order differential and filtering of the gray scale. Moreover, the extracted feature points are uniform and reasonable due to selecting the extreme point within an adjacent area. In addition, since Harris algorithm only uses the first-order derivate and Gaussian filtering window, it is a stable and well-performed algorithm of extracting feature point. It is also insensitive to image rotation, gray scale change, noise impact and viewpoint transform. However, Harris algorithm has too much data to calculate. And the larger the image is, the more the data calculations are. These massive data cost a great deal of time on CPU, which has been the bottleneck of processing performance for Harris corner detection algorithm. Consequently, it can't meet the requirement of real time which is increasingly emphasized in computer vision processing. Fortunately, all the calculations corresponding to each pixel are same and can be done in parallel, so we can transfer them to GPU in which the SIMD (Single Instruction and Multiple Data) architecture of the shader processor array is suitable and fast for such same calculations.

III. HARRIS ALGORITHM IMPLEMENTATION ON GPU

In the process of Harris algorithm implementation on GPU, general purpose GPU (GPGPU), OpenGL and GLSL are involved. The GPGPU technology means using the heterogeneous computing resources of GPU to implement a large scale of computations together with CPU. Classical GPGPU programmable pipeline is shown in Fig. 1. The shader processors in GPU consist of vertex shader processors and fragment shader processors: vertex shader

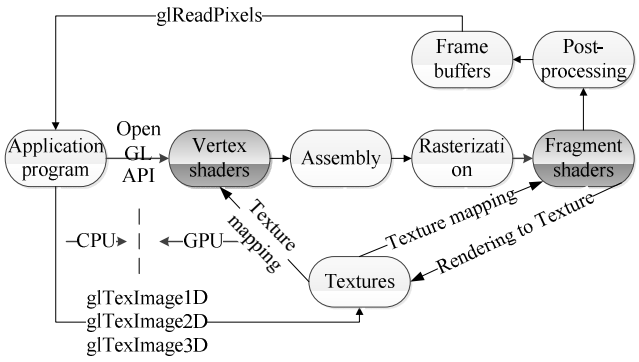

Figure 1. Classical GPGPU programmable pipeline.

processors deal with vertex data, while fragment shader processors deal with rasterized pixel data. In this paper fragment shader processors are mainly used to deal with the parallelized data. The parallelized data to be calculated are written by CPU into the texture buffer via generating texture using glTexImage1D, glTexImage2D or glTexImag3D, read in GPU by texture mapping in texture cache and processed via dozens of (or even hundreds of) vertex shaders or fragment shaders with certain calculation procedures. The data outputted from fragment shaders are rendered to the texture buffer which can be mapped by the next process stage again, or to the off-screen frame buffers which can be read by CPU. Post-processing consists of scissor test, stencil test, alpha test, blending and dithering, which are usually used in graphic processing.

Classical GPGPU depends on graphics application program interface (API) and the shading language. OpenGL is a large graphics API which can realize complex graphics effects. GLSL could be utilized to realize general-purpose computing on GPU. By using GLSL customized program can be written to replace the default graphics operations and realize customized graphics algorithms. Both OpenGL and GLSL have a platform-independent generality, so we can use them to realize Harris corner detection algorithm for portability rather than using CUDA, because CUDA has some limits to graphics card and GPU chips.

The implementation of Harris algorithm on GPU can be arranged as Fig. 2. The implementation starts from reading

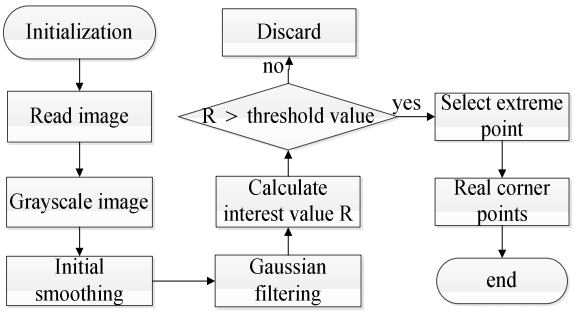

Figure 2. Program flow of Harris algorithm on GPU.

the color image and transfers the data to GPU by generating texture, uses glTexture2D to bind image data to texture0 and transforms the color image to gray scale image in fragment shader1, from where the outputted data are rendered to the texture1 in the external memory of GPU.

The second part actives the texture1 and reads it from the external memory to GPU, then calculates first-order gray scale gradient and executes the initial smoothing in fragment shader2. The outputted data are rendered back to texture2 for the next step of texture mapping.

The third part uses Gaussian window to filter the data of texture2 and calculates the interest values of pixels into texture3 with fragment shader3.

The fourth part is to judge the interest values in texture3 whether they exceed certain threshold value. If they do not, discard them; if they do, then save them and select the extreme point as the real corner point within an adjacent area. The data of real corner points cover the former data in texture3. Finally display the image with real corner points.

The load distribution between CPU and GPU is shown in Fig. 3. We can see four main parts of massive data processed on GPU are highlighted with black shadow filling. Generally, in the implementation process of algorithm on GPU, the data transmission between CPU and GPU should be reduced as much as possible in order to avoid a performance drop caused by waiting for transmission. In this implementation the data outputted from fragment shaders are rendered back to texture buffer for the next step of texture mapping rather than to the off-screen frame buffers, and the data used for the next step of texture mapping are read from the external memory of GPU rather than from CPU. Thus, it only needs to care the time consumption of data transmission between

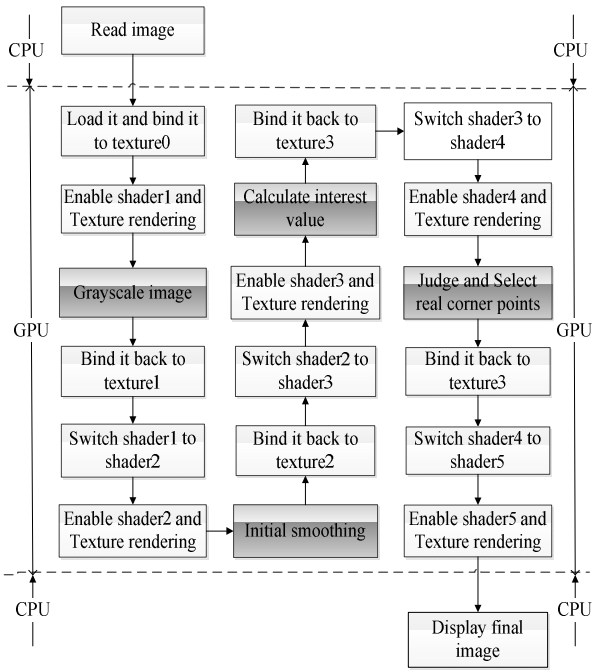

Figure 3. Optimized load distribution between CPU and GPU.

GPU and the external memory of GPU, which has a few impacts on final speedup result. Moreover, it achieves the maximum performance optimization for Harris corner detection algorithm implemented on GPU.

IV. EXPERIMENTS AND ANALYSIS

In order to verify the advantage of the implementation on GPU, we implement Harris corner detection algorithm by using OpenGL 2.0 on NIVIDIA GTX 650 that contains 384 shader processors with 80 GB/s memory bandwidth. Moreover, to do a comparison we also implement it with C language by using visual studio 2008 on Pentium Dual-Core 3.19 GHz CPU.

Three groups of experimental images are done with 5*5 Gaussian window by using the pixels of 162*122, 400*400 and 560*555 which are shown in Fig. 4. The images in the right column are the final images with the red corner points, while the images in the left column are source images. According to Harris algorithm flow described above, we measure the average execution time of five periods: the four are partial execution stages which are calculated completely by GPU because there are massive data to be processed in parallel, and the other is the full execution period of the algorithm. They are described in detail as the following:

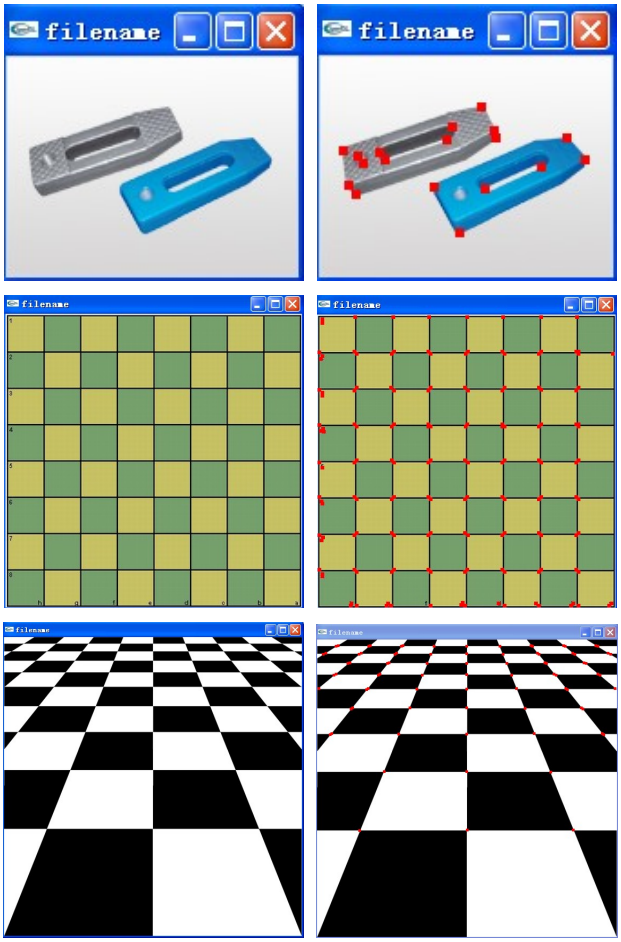

Figure 4. Example images used in experiments (the above are used in experiment 1, the middle are used in experiment 2 and the below are used in experiment 3).

Period 1: transformation of the source color image to gray scale image.

Period 2: initial smoothing.

Period 3: calculations of the interest values.

Period 4: judgment and the selection of extreme points within 3*3 window.

Period 5: the full execution process of the algorithm.

The time consumption comparison between the implementation on GPU and the implementation completely on CPU is shown in table I. In the three experiments, the execution times of the five periods are all reduced greatly by using GPU to accelerate the data calculations. The full execution process (Period 5) can achieve a speedup more than 73 times at most and 9 times at least. Moreover, the speedup of Period 5 is lower than Period 3, Period 4 and higher than Period 1 and 2 greatly. It is because Period 5 not only includes synchronization operations between CPU and GPU but also the data transfer between GPU and the external memory of GPU, which have a few impacts on the

TABLE I. TIME CONSUMPTION COMPARISON BETWEEN CPU AND GPU

		CPU time (ms)	**GPU time (ms)**	**CPU time /GPU time**
Experiment 1	Period 1	0.538	0.344	1.56
	Period 2	0.391	0.208	1.88
	Period 3	4.011	0.335	11.97
	Period 4	3.323	0.230	14.45
	Period 5	8.243	0.860	9.58
Experiment 2	Period 1	5.213	0.345	15.11
	Period 2	3.894	0.287	13.57
	Period 3	31.000	0.603	51.41
	Period 4	2.609	0.318	82.03
	Period 5	64.873	1.310	49.52
Experiment 3	Period 1	8.414	0.442	19.04
	Period 2	6.239	0.356	17.53
	Period 3	62.105	0.729	85.19
	Period 4	51.548	0.395	130.50
	Period 5	124.698	1.695	73.57

final speedup result. We can also see that the speedup of Period 3 and Period 4 is higher than Period 1 and 2 greatly,and the speedup of the large image is much higher than the small image. It is because the more the parallelization calculations are, the higher the utilization ratio of the hardware computing resources is.

V. DISCUSSIONS

In order to optimize the implementation on GPU, some factors should be taken into account. First of all, when the quantity of data reach a certain value and continue to increase, the speedup of algorithm implementation on GPU will slow down because of bandwidth limits between CPU and GPU. Thus, data transmission between CPU and GPU should be reduced as much as possible. Secondly, in order to avoid the communication bottleneck between GPU and the external memory of GPU, texture compression and decompression could be considered to reduce the transmission of texture data. Thirdly, it's important for performance optimization of algorithm implemented on GPU to well arrange what parts of algorithm are suitable for parallel processing on GPU and what parts of algorithm are suitable on CPU. What's more, the calculations involving associated pixels should be reduced to a minimum, since it may lead to a serious cache conflict and a low hit ratio of texture cache. The XY-type GPU cache index scheme could be used to solve such problem and improve the hit ratio of texture cache greatly [11]. Finally, dividing the image into blocks and storing them in block sequences could help to reduce the page miss in the external memory, because the storage structure of image data has an effect on the access efficiency of the external memory.

VI. CONCLUSION

Harris algorithm is a corner detection method, which has the advantage of robustness and accuracy. However, real time is a bottleneck since there are lots of data calculations. To solve this problem, this paper provides a way of implementation on GPU for embedded systems, which fully utilizes special parallel computing resources in GPU hardware to accelerate algorithm. It is realized with OpenGL and GLSL to meet the need of portability because both of OpenGL and GLSL have a platform-independent generality. Three group images of different sizes are experimented to prove the acceleration effects of algorithm implemented on GPU. The experiment results show that the execution performance of the implementation on GPU is over 73 times speedup of that completely on CPU at most and 9 times at least, and can meet the requirements of real time, accuracy and robustness. Finally, some aspects about optimization of algorithm implementation on GPU and the corresponding methods are briefly discussed.

ACKNOWLEDGMENT

This work was supported partly by Hunan Science and Technology Project (2010ck3010) and Natural Science Foundation of Hunan Province of China (12JJ6058).

REFERENCES

[1] Kitchen L, Rosenfeld A. Gray-level Corner Detection [J]. Pattern Recognition Letters, 1 (2): 95-102, 1982.

[2] Mokhtarian F, Suomela R. Robust Image Corner Detection through Curvature Scale Space [J]. IEEE Transaction on Pattern Analysis and Machine Intelligence. 20 (12): 1376-1381, 1998.

[3] Freeman H, Davis L S. A Corner Finding Algorihtm for Chain-coded Curves [J]. IEEE Transaction on Computers. C-26 (3): 297-303, 1977.

[4] Wang H, Brady M. Real-time Corner Detection Algorithm for Motion Estimation [J]. Image and Vision Computing. 13(9): 695-703, 1995.

[5] Harris C G, Stephens M J. A Combined Corner and Edge Detector. Proceedings of the 4th Alvey Vision Conference [C]. Manchester, England, 147-151, 1988.

[6] Guo Chenguang, Li Xianglong, Zhong Linfeng, Luo Xiang. A Fast and Accurate Corner Detector Based on Harris Algorithm [C]. Third International Symposium on Intelligent Information Technology Application. Nanjing, China, 49-51, 2009.

[7] Zhao Wanjin, Gong Shengrong, Liu Chunping, Shen Xiangjun. A Daptive Harris Corner Detection Algorithm [J]. Computer Engineering. 34 (10): 212-215, 2008.

[8] Zhang Xiaohong, Li Bo, Yang Dan. A Novel Harris Mullti-scale Corner Detection Algorithm [J]. Journal of Electronics&Information Technology. 29 (7): 1735-1738, 2007.

[9] Jun Zhang, Shuhua Luo, Xianru Liu. Weighted Mean Shift Object Tracking Implemented on GPU for Embedded Systems [C]. 2012 International Conference on Control engineering and communication technology (ICCECT). Shenyang, China, 7-9 December, 982-985, 2012.

[10] Xiao Han, Zhou Qinglei, Zhang Zuxun. Parallel Algorithm of Harris Corner Detection Based on Multi-GPU [J]. Geomatics and Information Science of Wuhan University. 37 (7): 876-881, 2012.

[11] Jun Zhang. Research on XY-type GPU Cache Index Mapping Scheme to Avoid Conflict Miss [C]. The 16th National Conference on Images and Graphics (NCIG 2012) and the 6th Workshop on 3D Images. Changchun, China, 8-10 August, 656-660, 2012.