Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/072L49

Fast Implementation of Scale Invariant Feature Transform Based on CUDA

Meng Lu[1]

[1]College of Information Science and Engineering, Northeastern University, China

Received: 16 Oct. 2012, Revised: 10 Jan. 2013, Accepted: 13 Jan. 2013
Published online: 1 Jun. 2013

Abstract: Scale-invariant feature transform (SIFT) was an algorithm in computer vision to detect and describe local features in images. Due to its excellent performance, SIFT was widely used in many applications, but the implementation of SIFT was complicated and time-consuming. To solve this problem, this paper presented a novel acceleration algorithm for SIFT implementation based on Compute Unified Device Architecture (CUDA). In the algorithm, all the steps of SIFT were specifically distributed and implemented by CPU or GPU, accroding to the step's characteristics or demandings, to make full use of computational resources. Experiments showed that compared with the traditional implementation of SIFT, this paper's acceleration algorithm can greatly increase computation speed and save implementation time. Furthermore, the acceleration ratio had linear relation with the number of SIFT keypoints.

Keywords: CUDA acceleration, Scale-Invariant feature transform, Image feature, Feature descriptor

1 Introduction

Scale-Invariant Feature Transform (SIFT) was published by Lowe in 1999 [1], and improved in 2004 [2], which was used to detect and describe local image features. SIFT is an excellent feature descriptor, because it is invariant to uniform scaling, orientation, and partially invariant to affine distortion and illumination changes. SIFT's applications include object recognition, robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, individual identification of wildlife and match moving.

Compute Unified Device Architecture (CUDA) is a parallel computing architecture developed by Nvidia for graphics processing. CUDA is the computing engine in Nvidia graphics processing units (GPU) that has ten times higher computation performance than CPU and is accessible to software developers through variants of industry standard programming languages. Due to CUDA, acceleration of large-scale computation became feasible. [3,5,9]. CUDA acceleration was widely used in image processing, Zhao and colleagues proposed a corresponding relation between the offset and the current viewpoint by establishing mathematics function of the detail image pixel coordinates and the height values [10]. Wu and colleagues proposed a GPU-aided parallel interpolation algorithm in order to scale video image real-timely [11]. CUDA acceleration was also widely used in biology sciences, Kim and colleagues proposed a method for computing the SES (solvent excluded surface) of a protein molecule in interactive-time based on GPU acceleration [12].

Implementation of SIFT is complicated and time consuming, which can't meet the real-time require of some applications, such as medical clinical application.To solve this problem, many scholars made attempts to accelerate SIFT implementation. Sinha and colleagues presented an acceleration algorithm of SIFT extraction based on OpenGL and Cg program language using Nvidia graphical card. However, they didn't complete all computation steps on GPU, which caused huge data transfer between GPU and CPU and cost much time [6]. Zhang and colleagues accelerated SIFT on intel 8 core CPU, and presented some optimization techniques to improve the implementation's performance on multi-core system [7], but this method can't widely spread, because it was too expensive for many applications.

This paper presented an acceleration algorithm of SIFT extraction based on CUDA, which made full advantage of GPU's abilities on float point computation,

* Corresponding author e-mail: menglu1982@gmail.com

parallel computation and memory management to optimize computational resources management and data transferring.

2 SIFT

2.1 Scale-space

Scale-space is a formal theory for handling image structures at different scales from physical and biological vision, by representing an image as a one-parameter family of smoothed images. The main type of Scale-space is the linear (Gaussian) Scale-space, which can be defined by formula(1):

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{1}$$

where $I(x,y)$ represented one image, * represented convolution, and $G(x,y,\sigma)$ represented Gaussian filter function.

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{2}$$

where (x,y) represented image coordinates, σ represented scale level. So, Difference of Gaussian (DoG) Scale-space can be defined as:

$$\begin{aligned} D(x,y,\sigma) &= (G(x,y,k\sigma) - G(x,y,\sigma))) * I(x,y) \\ &= L(x,y,k\sigma) - L(x,y,\sigma) \end{aligned} \tag{3}$$

2.2 Local Keypoint Detection

Once DoG images have been obtained, keypoints are identified as local minima/maxima of the DoG images across scales. This is done by comparing each pixel in the DoG images to its eight neighbors at the same scale and nine corresponding neighboring pixels in each of the neighboring scales. If the pixel value is the maximum or minimum among all compared pixels, it is selected as a candidate keypoint.

2.3 Keypoint Localization

Scale-space extreme detection produces too many keypoint candidates, some of which are unstable. The next step in the algorithm is to perform a detailed fit to the nearby data for accurate location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge.

The interpolation of keypoints is done using the quadratic Taylor expansion of the DoG scale-space function.

$$D(X) = D + \frac{\partial D^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X \tag{4}$$

Then, the location of the extreme $\hat{X}$, is determined by taking the derivative of this function with respect to X and setting it to zero.

$$\hat{X} = -\frac{\partial^2 D^{-1}}{\partial X^2} \frac{\partial D}{\partial X} \tag{5}$$

$$D(\hat{X}) = D + \frac{1}{2} \frac{\partial D^T}{\partial X} \hat{X} \tag{6}$$

The DoG function will have strong responses along edges, even if the candidate keypoint is not robust to small amounts of noise. Therefore, in order to increase stability, Hessian matrix is used to eliminate the keypoints that have poorly determined locations but have high edge responses.

$$\mathbf{H} = \begin{pmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{pmatrix}$$

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta \tag{7}$$

$$Det(H) = D_{xx} D_{yy} - (D_{xy})^2 = \alpha\beta \tag{8}$$

where α represented bigger eigenvalue, β represented smaller eigenvalue. Supposed that $\alpha = r\beta$, we can get formula (9).

$$R = \frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{9}$$

It follows that, for some threshold eigenvalue ratio r_{th}, if R for a candidate keypoint is larger than $(r_{th}+1)^2/r_{th}$, that keypoint is poorly localized and hence rejected.

2.4 Orientation Assignment

Each keypoint is assigned one or more orientations based on local image gradient directions. This is the key step in achieving invariance to rotation as the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.

$$\begin{aligned} m(x,y) = sqrt(L(x+1,y) - L(x-1,y))^2 + \\ (L(x,y+1) - L(x,y-1))^2 \end{aligned} \tag{10}$$

$$\begin{aligned} \theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/ \\ L(x+1,y) - L(x-1,y))) \end{aligned}$$

where $m(x,y)$ represented the gradient magnitude, $\theta(x,y)$ represented the orientation.

2.5 Keypoint Descriptor

First a set of orientation histograms are created on 4×4 pixel neighborhoods with 8 bins each. These histograms are computed from magnitude and orientation values of samples in a 16×16 region around the keypoint such that each histogram contains samples from a 4×4 sub-region

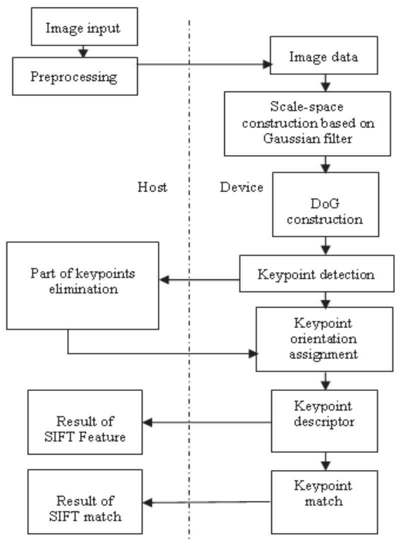

Fig. 1: Flow chart of CUDA-based acceleration of SIFT feature extraction algorithm

of the original neighborhood region. The magnitudes are further weighted by a Gaussian function with equal to one half the width of the descriptor window. The descriptor then becomes a vector of all the values of these histograms. Since there are $4 \times 4 = 16$ histograms each with 8 bins, the vector has 128 elements. This vector is then normalized to unit length in order to enhance invariance to affine changes in illumination.

3 CUDA-Based Acceleration of SIFT Computation

In this section, the whole details about SIFT acceleration method based on CUDA was presented. We made full advantage of GPU's abilities of float point computation, parallel computation, memory management, and give reasonable role to CPU, GPU in the SIFT computational procedure. Flow char of the SIFT acceleration algorithm based on CUDA was shown in Figure 1.

3.1 DoG Scale-space Construction

Firstly, image data was transferred from host memory to device memory and bound to texture memory. The reasons that we chose texture memory were: (1) Texture memory can access texture cache which was optimized for 2D array data in GPU, and have high performance under the circumstances of random access. (2) Texture memory can do bilinear interpolation in hardware to make acceleration of image processing. (3) While Gaussian filter was applied to 2D image array data, it's necessary to make judgment on array bounds. And CUDA program's not good at conditional statements. Therefore, texture memory was chosen, because it can efficiently and automatically solve this problem. Gaussian filter's parameters were computed in host based on formula (1), and results were transferred from host to device constant memory. Each image was processed in one grid. Each grid was divided into 16×16 blocks. Each block covered one part of the image, and was divided into 256 threads. Each thread covered one pixel and its 8 neighborhoods. Intermediate results of computation were saved in shared memory, and final result (Gaussian pyramid) was saved in global memory. Once Gaussian pyramid was obtained, DoG scale-space can be constructed based on formula (3). During DoG construction, the results were only correlated with adjacent layers in the same Gaussian pyramid group, and had no influences on the obtained Gaussian pyramid. Therefore, Gaussian pyramid can be divided into different blocks according to different groups and layers. And in each block, DoG scale-space was computed by formula (3).

3.2 Local Keypoints Detection

Suppose that Gaussian pyramid had O groups, each group had S layers. DoG had O groups, each group had $S-1$ layers. Correspoindingly, we can get O groups local keypoints, and each group had $S-3$ layers. Therefore, kernel functions can be circularly called O times to detect local keypoints, and each circulation was used to process image data in DoG for each scale. In the procedure of local extreme points detection, 26 neighborhood of each candidate pixel should be considered, which may greatly increase the amount of calculation. In the present paper, we firstly did extreme points detection in 8 neighborhood, experiments showed that[8,9,10] 95% candidate pixels can be eliminated. And then, we did extreme points detection in 26 neighborhood for the kept pixels. To a large extent, this method can improve calculation efficiency and save time. After local keypoints were preliminarily obtained, the result was transferred to host for further selection which has been discussed in section 2.3. Then the result of further selection was transferred back to device, and saved in global memory.

3.3 Keypoint orientation assignment

Each keypoint was processed in one block to calculate gradient orientation and magnitude by formula (10). Each block was divided into 16×16 threads, and each thread processed one pixel (shown in Figure 2).

3.4 Keypoint Descriptor

The present paper formulated the keypoint descriptor as 128 dimensional vector. It's similar to keypoints

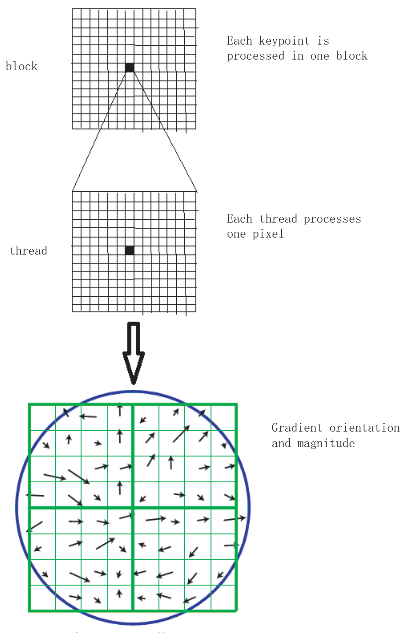
block
Each keypoint is processed in one block
thread
Each thread processes one pixel
Gradient orientation and magnitude

Fig. 2: Gradient orientation and magnitude of keypoints

orientation calculation that each vector was computed in one block, which was divided into 16×16 threads. All these 256 threads were divided into 4×4 sub-regions, and each sub-region was arranged into a histogram with eight bins based on the result of gradient orientation. Each thread calculated one pixel's degree of contribution to the histogram's bins.

4 Experiments and Results

Workstation's hardware and software used in the experiments was shown in table 1.

In the experiment, this paper's acceleration method based on CUDA was compared with standard SIFT[1,2]. We focused on the computational efficiency and computing time of two SIFT implementations. The standard SIFT was implemented in CPU. Test images were selected from BSDS500 database of UC Berkeley Computer Vision Group. To verify the correlation between the number of SIFT keypoints and computational efficiency, the comparisons were made under different image resolutions, which were $300 \times 200, 640 \times 480, 800 \times 640, 1200 \times 800, 1600 \times 1200$. The parameters of SIFT were set as: (1) the number of group was determined by image resolution; (2) 3 layer of each group; (3) threshold of DoG was 0.04; (4) threshold of principle curvature was 10; (5) sub-region of keypoint descriptor was 4×4.

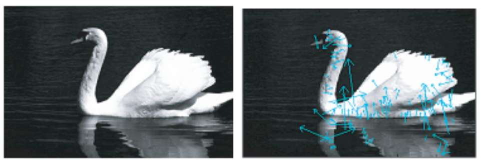

Fig. 3: Result of SIFT keypoints under the 300×200 image resolution, the number of detected keypoints by the present algorithm is 125

SIFT keypoints calculation results of different image resolutions were shown in Figure 3-Figure 7. The number of detected keypoints increased according to the image resolutions.

The comparison of computational efficiency between the present algorithm and the standard SIFT was shown in table 2. The numbers of detected keypoints of present algorithm and standard SIFT were almost the same under every image resolution, which meant that the present SIFT accelerated method could detect accurate keypoints. From table 2, we can conclude that higher the image resolution is, more SIFT keypoints can be obtained, and higher the present algorithm's speed-up ratio is.

The correlation between speed-up ration and the number of SIFT keypoints was shown in Fig. 8. The acceleration ratio approximatively had linear relation with the number of SIFT feature points.

Table 1: Computer's hardware and software used in the experiments

Items	Details
CPU	Intel Xeon X3440 2.53G2
Memory	DDR3 1333MHZ 8G
GPU	Nvidia Quadro 600
Number of multi-processor	2
Number of core in each processor	48
Video memory	1G DDR5
Thread maximum in each block	1024
Data transferring speed between host and device	3500M/s
Data transferring speed from device to device	7800M/s
Operation system	Win7 Professional

Table 2: Comparison between the present algorithm and the standard SIFT under different image resolutions

Items	Image A	Image B	Image C	Image D	Image E
Image resolution	300 × 200	640 × 480	800 × 640	1200 × 800	1600 × 1200
The number of keypoints by the standard SIFT	127	1746	3186	5761	8672
Time used by the standard SIFT (ms)	365	1609	2815	5260	9053
The number of keypoints by the present algorithm	125	1781	3175	5736	8677
Time used by the present algorithm (ms)	135	311	361	426	463
Speed-up ratio	2.70	5.16	7.78	12.36	19.54

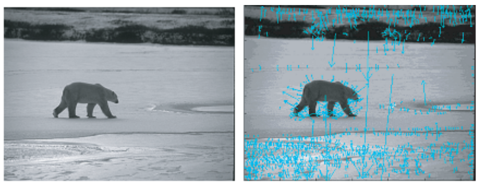

Fig. 4: Result of SIFT keypoints under the 640 × 480 image resolution, the number of detected keypoints by the present algorithm is 1781

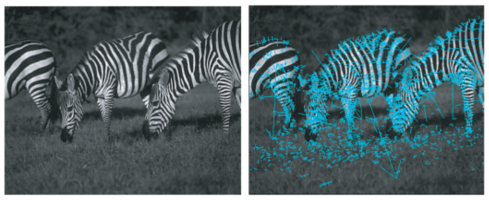

Fig. 5: Result of SIFT keypoints under the 800 × 640 image resolution, the number of detected keypoints by the present algorithm is 3175

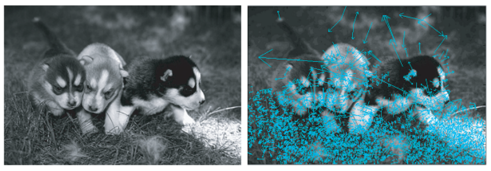

Fig. 6: Result of SIFT keypoints under the 1200 × 800 image resolution, the number of detected keypoints by the present algorithm is 5736

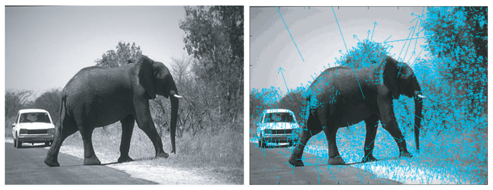

Fig. 7: Result of SIFT keypoints under the 1600 × 1200 image resolution, the number of detected keypoints by the present algorithm is 8677

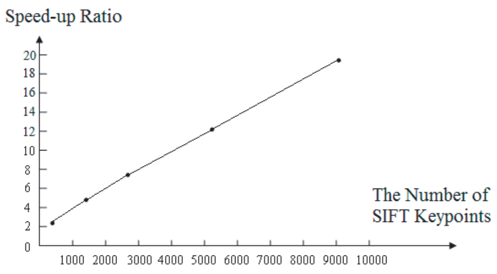

Fig. 8: The correlation between speed-up ration and the number of the SIFT keypoints of the present algorithm, the acceleration ratio approximatively had linear relation with the number of SIFT feature points

5 Conclusion

This paper presented a CUDA-based acceleration algorithm of SIFT feature extraction. The present algorithm can greatly enhance SIFT computational speed, and the speed-up ratio linearly increased with the increase of the number of SIFT keypoints.

Acknowledgement

This research was supported by National Natural Science Foundation of China (61101057).
The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] Lowe D. object recognition from local scale-invariant features[J], International Conference on Computer Vision(ICCV), 1999:1150-1157.

[2] Lowe D. Distictive Image Features From Scale-Invariant Keypoints[J], International Journal of Computer Vision, 2004,60(2):91-110.

[3] Nian H. GPGPU and Image Matching Parallel Algorithm Based on SIFT[D], XiDian University, 2010.

[4] Wang R, Liagn H, Cai Xuan-ping. Study of SIFT Feature Extraction Algorithm Based on GPU[J], Modern Electronics Technique, 2010,15:41-43.

[5] Zuo HR, Zhang QH, Xu Y. Fast Sobel Edge Detection Algorithm Based on GPU[J], Opto-Electronic Engineering, 2009,36(1):41-43.

[6] Sinha S, Frahm J M, Pollefeys M. Feature Tracking and Matching in Video Using Programmable Graphics Hardware[J], Available from: http://www.springerlink.com/content/5rv615p24360/.

[7] Zhang Qi,Chen Y, Zhang YM. SIFT Implementation and Optimization for Multi-core Systems. IEEE International Symposium on Parallel and Distributed Processing, 2008. p. 1-8.

[8] Tian W, Xu F, Wang HY, Zhou B. Fast Scale Invariant Feature Transform Algorithm Based on CUDA[J], Computer Engineering, 2010,36(8):219-221.

[9] Gan XB, Wang ZY, Shen L, Zhu Q. Data layout pruning on GPU, Applied Mathematics Information Sciences,2011,5(2):129-138.

[10] Zhao CJ, Wu HR, Gao RH. Realistic and detail rendering of village virtual scene based on pixel offset, Applied Mathematics Information Sciences, 2012:6(3):769-775.

[11] Wu TH, Bai BG, Wang P. Parallel catmull-rom spline interpolation algorithm for image zooming based on CUDA, Applied Mathematics Information Sciences, 2013,7(2):533-537.

[12] Kim B, Kim KJ, Seong JK. GPU accelerated molecular surface computing, Applied Mathematics Information Sciences, 2012,6(1):185-194.

Meng Lu received the MS degree in Computer Science from Northeastern University in 2007, and received the PhD degree in Computer Science from the Northeastern University in 2010. He is currently a lecturer in Northeastern University. His research interests are in the areas of computer vision, medical image processing, and 3D visualization. He has published research articles in reputed international journals of mathematical and engineering sciences.