GPU-BASED FAST SCALE INVARIANT INTEREST POINT DETECTOR

Hongtao Xie, Ke Gao, Yongdong Zhang, Jintao Li, Yizhi Liu

Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China
xiehongtao@ict.ac.cn

ABSTRACT

To take full advantage of the powerful computing capability of graphics processing units (GPU) to speed up local feature detection, we present a novel GPU-based scale invariant interest point detector, coined Harris-Hessian(H-H). H-H detects Harris points in low scale and refines their location and scale in higher scale-space with the determinant of Hessian matrix. Compared to the existing methods, H-H significantly reduces the pixel-level computation complexity and has better parallelism. The experiment results show that with the assistance of GPU, H-H achieves up to a 10-20x speedup than CPU-based method. It only takes 6.3ms to detect a 640 × 480 image with high detection accuracy, meeting the need of real-time detection.

Index Terms— interest points, scale invariance, local features, Graphics Processing Units (GPU), Compute Unified Device Architecture (CUDA)

1. INTRODUCTION

Local features have been demonstrated to be well adapted to recognition and matching as well as many other applications, as they are robust to content changes and geometric deformations. For all these transformations, special attention should be paid to scale change [1]. Lowe [2] claimed that additional complexity of full invariance to all the changes would have a negative impact on the robustness of the features. Therefore, we focus on scale invariant detector.

Features that have proved to be especially appropriate for representing the image are interest points. Harris interest point detector [3] has the best performance among all other detectors [4], but it is not invariant to scale transformation. Mikolajczyk [5] proposed Harris-Laplacian (H-L) detector. The H-L is scale invariant, however, it is time-consuming. Herbert [1] created a new detector, which they coined SURF. SURF consumes less time than H-L, but it still cannot meet the need of real-time application. Researchers have been trying to speed up the extraction of local features for real-time detection. Due to the limited computing ability of CPU, the goal has not been achieved.

In the past few years, the progress of GPU is tremendous. The computational capability of GPU today is much higher than that of the CPU. Due to its powerful computing capability, the GPU nowadays serves not only for graphics display, but also for general-purpose computation, such as molecular dynamics and image processing. NVIDIA recently announced a powerful GPU architecture called “Compute Unified Device Architecture” (CUDA) [6]. It largely improves the programmability of GPU and can efficiently map a computing problem onto the GPU. Thus, we propose a GPU-based scale invariant interest point detector.

Although a few GPU-based local feature extraction methods have been proposed [7-9], they just implemented the original algorithms on GPU. The original algorithms are complex and need a lot of data copy between different memory spaces, which become a performance bottleneck in CUDA [6]. The great computational complexity is caused by multi-scale simulation. In this paper, we propose a new scale invariant interest point detection algorithm according to the characteristics of GPU and CUDA. To avoid multi-scale simulation, the new algorithm detects interest points with Harris detector in low scale and refines the points in higher scale-space relying on the determinant of Hessian matrix. It has good parallelism and can satisfy the demand of real-time detection under large scale data with high detection accuracy.

The paper is organized as follows. Section 2 describes the new algorithm. Section 3 presents the implementation of the algorithm on GPU. Finally, section 4 shows the experiment results and section 5 concludes the paper.

2. GPU-BASED SCALE INVARIANT INTEREST POINT DETECTOR

To obtain scale invariant detector, Mikolajczyk [5] combined the Harris detector with the Laplacian-based (LoG) scale selection. The LoG representation has a drawback that is local maxima may be detected in the neighborhood of contours or straight edges, where the signal change is only in one direction. Therefore, false detection exists. The maxima of the determinant of the Hessian matrix penalize points for which the second derivatives detect signal changes in only one direction. So, using the determinant of Hessian matrix can deal with false detection effectively. Herbert [1] also pointed out that adoption the determinant of the Hessian matrix rather than its trace (LoG) seemed advantageous.

We present a novel GPU-based scale invariant interest point detector—Harris-Hessian (H-H). H-H first detects Harris points in the low scale of the image, to obtain the candidate points. Then it applies the determinant of Hessian matrix to eliminate the false detection and confirm the scale of the exact interest points in higher scale-space of the image.

2.1 Low scale Harris point detection

We first detect Harris points in low scale σ_0 to get the candidate point set C_i . σ_0 can be equal to 0, which means detecting corners in the original image. σ_0 may choose different value in a small range of sale, such as [0, 1.2].

When detect directly in the original image, Harris detector [3] can be used; when σ_0 values in an interval, scale adapted Harris function proposed in [5] can be adopted, and the repeated points in C_i should be removed.

2.2 False detection elimination & scale confirmation

After obtaining C_i , for each point we apply the *DET* value of the Hessian matrix to remove the false detection and confirm the scale of interest points simultaneous. Given a pixel $X = (x, y)$ in an image I, the Hessian matrix $H(X,\sigma)$ in X at scale σ is defined as follows:

$$H(X,\sigma)=\begin{bmatrix} L_{xx}(X,\sigma) & L_{xy}(X,\sigma) \\ L_{xy}(X,\sigma) & L_{yy}(X,\sigma) \end{bmatrix}, \quad (1)$$

where $L_{xx}(X,\sigma)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2}g(\sigma)$ with the image I in pixel X , and similarity for $L_{yy}(X,\sigma)$ and $L_{xy}(X,\sigma)$.

The *DET* value is:

$$DET(H)=\left|L_{xx}\times L_{yy}-(0.9L_{xy})^2\right| \quad (2)$$

The steps of false detection elimination and scale confirmation are:

1. For each candidate point m in C_i , calculate its *DETs* under a range of scale σ_n .

   $\sigma_n = 1.2^n\sigma_0 (n = 1,2,...,N), \sigma_0 = 1.2$. The value of N is determined by the specific applications.
2. If the *DETs* of m attain no extremum in a series of σ_n , we reject it.
3. If m has maximum *DET* value in scale σ_n and that *DET* is greater than a threshold, then m is an interest point and its scale is σ_n .

$$DET(X,\sigma_n) > DET(X,\sigma_{n-1}) \wedge DET(X,\sigma_n) > DET(X,\sigma_{n+1})$$
$$DET(X,\sigma_n) > T \quad (3)$$

As shown in Fig.1, m has a comparison in scale-space created by the adjacent scales σ_{n-1} , σ_n and σ_{n+1} .

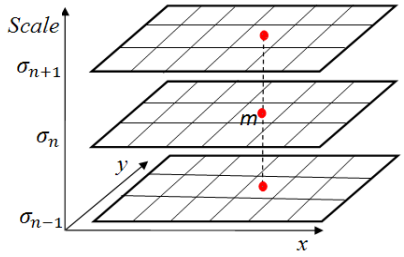

Fig. 1 Comparison in adjacent scale-space

2.3 Performance analysis of Harris-Hessian

Compared to [1, 5], the algorithm designed in this paper has the following three advantages:

1. Computational complexity is reduced significantly.
   [1, 5] need to do convolution and derivative operation in each scale of the image to detect interest points. [1, 5] also have to calculate each pixel's response to remove false detection points and confirm scale. Their computational complexity is $O(n^2)$. However, we only focus on the candidate points detected in low scale and the computational complexity is $O(n)$.
2. Hessian matrix is applied to reduce false detection.
   The determinant of Hessian matrix is more robust to the impact caused by noise, contours and straight edges than LoG.
3. Our method has better parallelism and is more suitable for GPU implementation.
   As [1, 5] need complex calculation in each scale and the capacity of shared memory and register is limited, they need several times of kernel invocation and data copying between different memory spaces for executing on GPU. It is not adapted to the architecture of GPU [6]. Our method only needs a few times of kernel invocation and data transfer. So it makes full use of shared memory, resulting in good performance.

3. IMPLEMENTATION ON GPU

3.1 Data partition and storage

As H-H only involves convolution and differential operation, we partition the source image equally into data-blocks and distribute them among the thread blocks, as illustrated in Fig.2. In our realization, there are 256 threads in a thread block, so the total number of thread block is:

$$Block\_num=\frac{image\_width}{16}\times\frac{image\_height}{16} \quad (4)$$

To speed up the process of the convolution, we approximate the Gaussian with box filters, which is beneficial to GPU acceleration too. As box filters are constant, we put them in the constant memory when the program starts to reduce the frequency of data transfer.

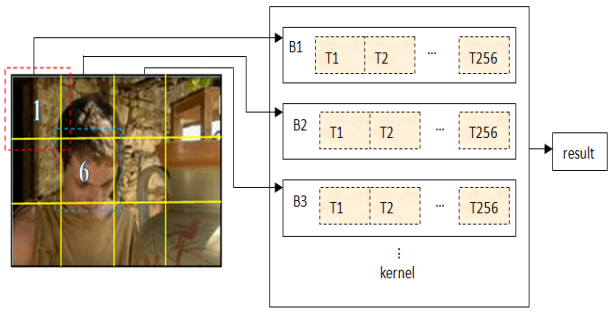

Fig. 2 Data partition

During the calculation of convolution, the problem of "boundary cases" will be confronted. As shown in Fig.2, when calculating convolution for blocks 1 and 6, the required data are represented by the red and blue dashed box. If we set "conditional statement" for each pixel, execution efficiency will be reduced [6]. So we employ texture memory, which handles the "boundary cases" automatically.

3.2 Execution flow

It takes two steps to calculate convolution. First, one kernel is invoked for the calculation of x direction; then the other kernel is called for the calculation of y direction, based on the previous result. In the process of false detection removal and scale confirmation, one kernel is called. If there are k candidate points, one thread block contains 64 threads and the num of thread block is $k/64$.

The execution flow of H-H on GPU is shown in Fig.3. Firstly, the data of source image and box filters are copied into texture memory and constant memory; then each thread gets its data and starts executing; the intermediate results are stored in shared memory, through which the threads in a thread block coordinate with each other; and then all threads continue executing until the task is completed; at last, the result is transferred to the DRAM.

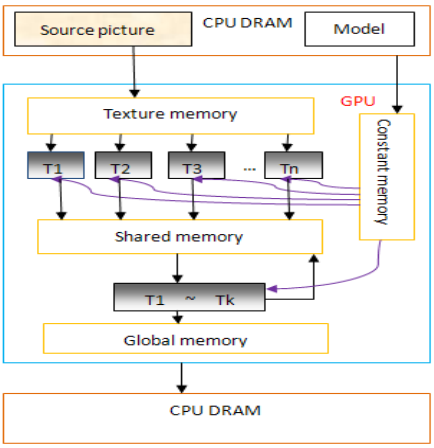

Fig. 3 Execution flow of H-H on GPU

4. EXPERIMENTS

The experiment environment is: Intel Core E8400 3.0GHz with 2048MB memory, NVIDIA GeForce 9800GTX+ with 512MB DRAM, Microsoft Windows XP sp2, CUDA Toolkit 2.1 and CUDA Driver (181.20).

To evaluate the performance of H-H detector, we develop an image copy detection system. We first extract H-H interest points; then compute SIFT descriptor [2] for each point at its scale. If the number of similar points between two images is greater than a threshold, they are matched.

Our dataset includes 11250 images, which include nature, people, news, video frame, drawing and cartoon. Their size ranges from 256×364 to 1024×1024. To construct test images, we choose 150 images from the dataset and add attack such as crop, contrast change, occlusion and zoom to them.

Fig.4 shows the time cost of GPU and CPU to detect the H-H points in images of different size. Compared to CPU, the GPU realization achieves up to a 10-20x speedup. It only takes 6.3ms to detect a 640×480 image, getting 437 points. It demonstrates the good parallelism of H-H.

Table 1 illustrates the precision of H-H to various kinds of attack. Attack rate refers to the strength of attack, such as "crop 20%" means that any 20% part of the original image is cut. Fig.5 presents an original image and some attacked images. The results show that our detector is very effective.

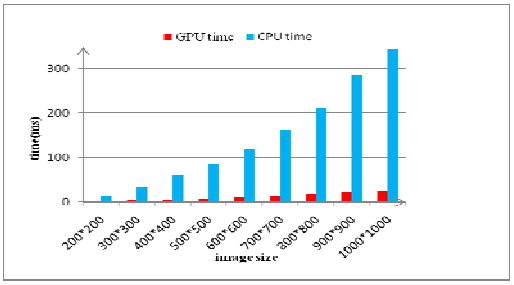

Fig. 4 Time cost of CPU and GPU to detect H-H points

Attack	Attack rate	Precision
crop	20%	93.5%
	30%	90.6%
contrast	30%	100%
	60%	96.8%
occlusion	30%	100%
	60%	96.8%
intensity	50%	100%
translation	10%	100%
	30%	87.5%
hue saturation	60%	100%
	180%	100%
zoom	2x	100%
add text		100%

Table 1. Precision of H-H to different attack

Fig. 5 The original image and transformed images

In order to test the effect of DET in false detection elimination and scale confirmation, we compare it with LoG [5] in the same system and dataset, as illustrated in table 2. The results prove that using DET makes the intertst pionts more representive .

	Precision	Recall
DET	95%	85.3%
LoG	93.4%	77.6%

Table 2. The performance comparison between DET and LoG

We also implement H-L [5] and SURF [1] on GPU with CUDA, and compare them with H-H in time performance and detection accuracy, to validate the effectiveness of H-H. Fig.6 shows the time property of the three algorithms. We can see that H-H costs significantly less time than the others. The advantage is more obvious in large-size images.

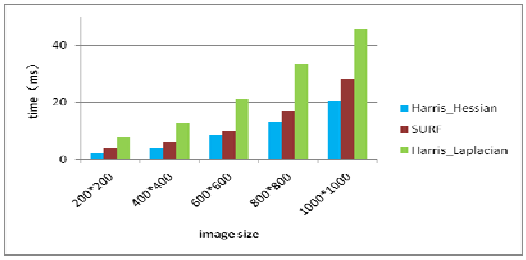

Fig. 6 Detection time comparison of three methods on GPU

Table 3 shows the detection accuracy of the three algorithms in our image copy detection system under the same data. H-L has the same descriptor as H-H and SURF is implemented exactly as explained in [1]. We could see that H-H maintains high accuracy.

	Precision	Recall
Harris-Hessian	95%	85.3%
Harris-LoG	95.7%	86.1%
SURF	90.4%	63.8%

Table 3. The accuracy of three algorithms

Note that [9] has implemented SURF on GPU in a different way with [1]. We compare our GPU-based H-H with [9] too. We find that the time performance of the two algorithms is comparative, but H-H detects fewer points than [9] and has better accuracy.

All the experiment results certify that GPU-based H-H significantly reduces the time consumption while maintaining high detection accuracy.

5. CONCLUSIONS

In this paper, we design a new GPU-based scale invariant interest point detector, according to the property of GPU and CUDA. To reduce computational complexity and the times of data copy, we detect interest points with Harris detector in low scale and refine the points in higher scale-space relying on the determinant of Hessian matrix. Experiment results show that with GPU, our algorithm can be 10-20 times faster than the CPU only configuration and maintains high detection accuracy. It outperforms the current state-of-the art, and can meet the need of real-time detection under large scale data. Our future work is to port the other part of copy detection, such as indexing and matching on the CUDA GPU.

6. ACKNOWLEDGEMENTS

This work is supported by the National Basic Research Program of China (973 Program, 2007CB311100);National High Technology and Research Development Program of China (863 Program, 2007AA01Z416);National Nature Science Foundation of China (60873165 60802028);Beijing New Star Project on Science & Technology(2007B071);Co-building Program of Beijing Municipal Education Commission

7. REFERENCES

[1] H. Bay, T.Tuytelaars, and L.Van Gool. “SURF: Speeded Up Robust Features”. *ECCV*, pp. 404–417, 2006.

[2] Lowe D. “Distinctive Image Features from Scale Invariant Keypoints”. *IJCV*, pp.91-110, 2004.

[3] C.Harris. “A Combined Corner and Edge Detector”. *Proceedings of 4th AVC*, pp.147-151, 1988.

[4] Mikolajczyk K, Tuytelaars T, et al. “A comparison of affine region detectors”. *IJCV*, pp.43-72, 2006.

[5] Mikolajczyk K, Schmid, C, et al. “Scale and affine invariant interest point detectors”. *IJCV*, pp. 63–86, 2004.

[6] NVIDIA.“NVIDIA CUDA Programming Guide version 2.0”. http://www.nvidia.com/object/cuda_get.html.

[7] S. Heymann, et al. “Sift Implementation and Optimization for General-Purpose GPU”. *Proceedings of the 15th WSCG*, 2007.

[8] S.Sinha, J. Frahm. “GPU-based Video Feature Tracking and Matching”. *Workshop on EDGE*, 2006.

[9] N. Cornelis and L.Van Gool.“Fast scale invariant feature detection and matching on programmable graphics hardware”. *Workshop on CVPR*, 2008.