Presentation date: March, 12, 2010 Paper number: 256-2

GPU implementation of ESM tracking

◯ Chuantao Zang Yoshihide Endo Koichi Hashimoto

Graduate School of Information Sciences, Tohoku University
{chuantao, endo, koichi}@ic.is.tohoku.ac.jp

***Abstract*— This paper describes our novel work of using graphic processing unit (GPU) on visual tracking. In this paper, we present our novel implementations of GPU based Efficient Second-order Minimization (GPU-ESM) algorithm. By utilizing the tremendous parallel processing capability of modern graphic hardware, we obtain significant processing acceleration from GPU over its CPU counterpart. Currently our GPU-ESM algorithm can process tracking area of 360×360 pixels at 145 fps on NVIDIA GTX295 board and Intel Core i7 920, which is approximately 30 times faster than CPU implementation. This speedup substantially improves the realtime performance of our system. In this paper, translation details of ESM algorithm from CPU to GPU implementation and novel optimizations are presented. The effectiveness of our GPU-ESM tracking algorithm is validated with experimental data.**

Key Words: ESM method, GPU, optimization.

I. INTRODUCTION

Visual tracking consists in the problem of estimating the incremental transformations which align a reference image with successive frames of a video sequence. As a fundamental task to many applications, general solutions can be classified into feature-based and direct (intensity-based) methods [1]. Among various visual tracking systems, one category has been designed to exploit the Cartesian information from the homography between two images of a planar object.

Usually homography solution is a typical minimization problem of sum of squared differences (SSD) between a region in reference image and a warped region in current image [2]. Many nonlinear optimization approaches have been proposed to deal with this least square optimization problem with different kinds of approximations, such as Standard Newton method, Gauss-Newton approximation [3]. Among these solutions, the efficient second-order minimization (ESM) algorithm is an elegant idea which obtains the same convergence speed as standard Newton method without computing the computational costly Hessian matrix [4]. By performing second order approximation of the SSD problem with only first order derivative, this method can get high convergence rate and avoid local minima close to the right global one.

However, when considering realtime visual tracking system, the main requirements of the tracking algorithms are efficiency, accuracy and stability. In Malis's paper [2], only the stability and convergence rate of ESM algorithm are evaluated with experiments. As far as we know, no data of processing speed is mentioned. From our experience, with the increase of tracking area size, for example, size of 300×300 pixels, the ESM computation still costs too much time and induces a relative low processing speed. To solve these problems and improve the performance of homography-based visual tracking system, We present GPU based ESM tracking algorithm (we call it GPU-ESM) to address the need for faster visual tracking algorithms. After several novel optimizations on GPU code, we succeed in achieving substantial acceleration over CPU ESM implementation. For tracking area of 360×360 pixels, our GPU-ESM can work at 145 fps, approximately 30X faster than CPU. This allows for a realtime visual tracking system with a higher speed camera. The rest of this paper is organized as follows. Section II reviews the relative works about ESM algorithm. Section III introduces the translation details of GPU-ESM so as to fully utilize the parallel architecture of GPU. Section IV describes the experimental results to validate our proposed system. Section V concludes this paper.

II. RELATED WORKS

ESM algorithm was first proposed by Dr. Malis in 2004 [5]. By performing second order approximation of the least square problem with only first order derivative, it shows great application potential in visual tracking. It has been reported in many different kinds of applications, such as visual tracking of planar object [2] and deformable object [6]. To our knowledge, there is no reports of GPU based ESM yet. For simplicity, we review

the main idea of homography-based visual control scheme with ESM algorithm [2].

First we suppose the planar object to track is projected in the reference image I^* with some "Template" region of q pixels. Tracking the reference template in the current image I consists in finding the homography transformation $\overline{G}$ that transforms each pixel P_i^* of the reference pattern into its corresponding pixel in the current image I, i.e. finding the homography $\overline{G}$ such that $\forall i \in \{1,2,...q\}$:

$$I(w(\overline{G})(P_i^*)) = I^*(P_i^*) \quad (1)$$

Suppose that we have an approximation $\widehat{G}$ of $\overline{G}$, the problem consists in finding an incremental transformation $G(x)$ (where the 8×1 vector $\mathbf{x}$ contains a local parameterization such that the difference between the region of the image I (transformed with the composition $w(\widehat{G}) \circ w(G(x))$ and the corresponding region in the image I^* is null. Tracking consists in finding the vector x such that $\forall i \in \{1,2,...q\}$, the image difference

$$y_i(x) = I(w(\widehat{G}) \circ w(G(\mathbf{x}))(P_i^*)) - I^*(P_i^*) = 0 \quad (2)$$

Let $\mathbf{y}(\mathbf{x})$ be the $q \times 1$ vector containing the image differences:

$$\mathbf{y}(\mathbf{x}) = [y_1(\mathbf{x}), y_2(\mathbf{x}), ...y_q(\mathbf{x})]^T \quad (3)$$

Therefore, the problem consists in finding $\mathbf{x} = \mathbf{x}_0$ verifying:

$$\mathbf{y}(\mathbf{x}_0) = \mathbf{0} \quad (4)$$

We linearize the vector $\mathbf{y}(\mathbf{x})$ around $\mathbf{x} = \mathbf{0}$ using the second-order Taylor series approximation:

$$\mathbf{y}(\mathbf{x}) = \mathbf{y}(\mathbf{0}) + \mathbf{J}(\mathbf{0})\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{H}(\mathbf{0})\mathbf{x} + \mathbf{O}(\|\mathbf{x}\|^3) \quad (5)$$

where $\mathbf{J}(\mathbf{0})$ and $\mathbf{H}(\mathbf{0})$ are the Jacobian matrix and Hessian matrix at $\mathbf{x} = \mathbf{0}$, seperately. In ESM algorithm, the Hessian matrices of vector $\mathbf{y}(\mathbf{x})$ are replaced by first-order Taylor Series approximation of vector $\mathbf{J}(\mathbf{x})$ about $\mathbf{x} = \mathbf{0}$:

$$\mathbf{J}(\mathbf{x}) = \mathbf{J}(\mathbf{0}) + \mathbf{x}^T\mathbf{H}(\mathbf{0}) + \mathbf{O}(\|\mathbf{x}\|^2) \quad (6)$$

Then equation (5) becomes

$$\mathbf{y}(\mathbf{x}) \approx \mathbf{y}(\mathbf{0}) + \frac{1}{2}(\mathbf{J}(\mathbf{0}) + \mathbf{J}(\mathbf{x}))\mathbf{x} \quad (7)$$

For $\mathbf{x} = \mathbf{x_0}$, we have

$$\mathbf{y}(\mathbf{x_0}) = \mathbf{y}(\mathbf{0}) + \frac{1}{2}(\mathbf{J}(\mathbf{0}) + \mathbf{J}(\mathbf{x_0}))\mathbf{x_0} = \mathbf{0} \quad (8)$$

With some mathematical proof by Dr. Malis [2], the sum of Jacobian matrix $\frac{1}{2}(\mathbf{J}(\mathbf{0}) + \mathbf{J}(\mathbf{x_0}))$ can be written as one matrix $\mathbf{J_{esm}}$. Therefore, for $\mathbf{x} = \mathbf{x_0}$, we have

$$\mathbf{y}(\mathbf{x_0}) = \mathbf{y}(\mathbf{0}) + \mathbf{J_{esm}}\mathbf{x_0} = \mathbf{0} \quad (9)$$

The solution $\mathbf{x_0}$ can be obtained by:

$$\mathbf{x_0} = \mathbf{J_{esm}^+}\mathbf{y}(\mathbf{0}) \quad (10)$$

The homography $\overline{G}$ can be calculated with this $\mathbf{x_0}$ with Lie Algebra operation. By obtaining this homography $\overline{G}$, visual tracking is accomlished.

III. System implementation

Our GPU implementations of ESM algorithms are carried out on a desktop with Intel Core i7-920 2.67GHz CPU, 3GB RAM and a NVIDIA GTX295 graphic board. The GTX295 board integrates two GTX280 GPUs inside and has 896MB GPU RAM for each GTX280 GPU. The interface between motherboard and GTX295 board is PCI-E x16 bus. Operating system is Windows XP (sp2) with NVIDIA's CUDA (Compute Capability 1.3).

For our GPU-ESM algorithm, we categorize our CUDA kernels in 6 categories.

1). Warping. This kernel completes the task that warping pattern image to current image with a known homography.

2) Gradient. This kernel calculates the image intensity gradient in X and Y directions.

3) J_{esm}. This kernel calculates the J_{esm} matrix in ESM algorithm.

4) Solving. This kernel finds solution X of linear equations

$$J_{esm}X = y(0) \quad (11)$$

5) Updating. This kernel updates homography with solution X from Solving kernel.

6) Correlation. This kernel calculates the correlation of warped area and pattern image to decide when to stop the ESM processing. As ESM algorithm is iterative minimization method, we have to set such threshold to stop the loop.

IV. Optimization and experiments

In this section, we share our optimization experience during our GPU-ESM implementation. Then we evaluate the whole performance of our GPU-ESM algorithm with experimental data.Though CUDA uses C language with several extensions which makes it easier than other GPU languages for programmer to write GPU code, to

make GPU code highly proficient, carefully optimization must be exploited and several important factors must be considered.

A. Memory optimization

Memory optimization mainly consists in using different kinds of GPU memory to accelerate the application. CUDA provides a hierarchy of memory resources. Among them, commonly used are register, shared memory, global memory and texture memory.

In our GPU-ESM, we intensively utilize the fast shared memory instead of the long-latency global memory. In kenel J_{esm}, the compute of J_{esm} matrix needs several intermediate results based on the image gradient. So we first load the image gradient data into shared memory of each block and then continue other computation on them. By using this "cache" like strategy, we reduce this kernel's processing time from 300us to 50us.

We also use texture memory in kernel "warping" because CUDA provides such bilinear filter function. We only need to set the filter mode to bilinear. When fetching the texture memory, the returned value is computed automatically based on the input coordinates. Though the running time is similar to our own kernel but with it we can skip its programming.

B. Memory coalescing

By memory coalescing a half warp of 16 GPU threads can finish 16 global data fetching in as few as 1 or 2 transactions. In our application, we also intensively use this technique. For example, to calculate the mean $\bar{I}$ in ZNCC correlation, first we need to calculate the sum of I. We use 1 block of 512 threads to accumulate all the 90000 pixels. The number of data processed in each block is at least $90000/512 \approx 175$. The normal idea is using "for-loop" in each thread like this:

$$for(k = threadeID; k < threadeID + 175; k++)$$

$$sum+ = I[k];$$

To use memory coalescing, we change the code to follows:

$$for(k = threadeID; k < 90000; k+ = 512)$$

$$sum+ = I[k];$$

Though both "for-loops" have same performance for a CPU thread, speedup really happens on GPU. GPU memory is accessed in a specific block mode, i.e. each GPU memory access will load data from a block of continuously addressing memory space. For example, it can load $I[0] \sim I[15]$ simultaneously by 16 GPU threads. In latter method, the loaded 16 data can be parallel processed by 16 GPU threads. Meanwhile, in former method, only one data is used in one thread while all other data is deserted. For each of other threads, they must invoke their own GPU memory access to fetch the one they need. Therefore using memory coalescing strategy, we can substantially reduce the total number of memory access. As GPU memory access belongs to the long-latency global memory access (several hundred GPU cycles), reducing global memory access provides us with great performance gain.

C. Experiments

With above optimizations, finally we realize our GPU-ESM and conduct experiments to compare with CPU-ESM on the same desktop. Processing frame rates are shown in Table I and Fig. 1.

TABLE I
GPU FPS AND CPU FPS

Size	M	CPU fps	GPU fps
32×32	1024	703	306
40×40	1600	479	304
64×64	4096	187	301
96×96	9216	86	287
128×128	16384	48	271
160×160	25600	32	254
192×192	36864	22	234
224×224	50176	16	215
256×256	65536	11	195
300×300	90000	8.2	172
360×360	129600	4.8	145

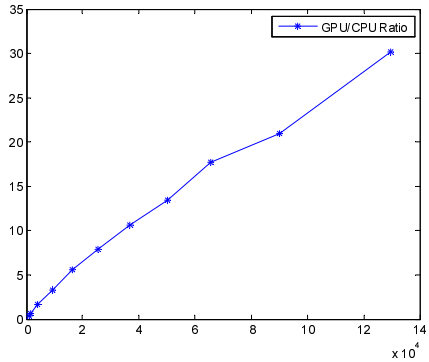

Fig. 1. GPU/GPU Ratio repecting to M

The data shows that using GPU can greatly accelerate the ESM algorithm. Meanwhile, as the $GPU/CPU\ Ratio$ increases with the tracking area size M, it also shows that GPU is more preferable for highly parallel processing. For those algorithms which can not be parallelized, using GPU

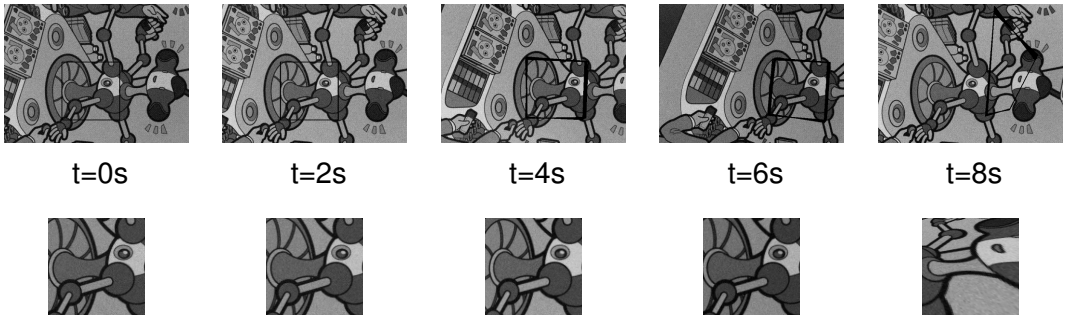

Fig. 2. ESM tracking results on CPU.

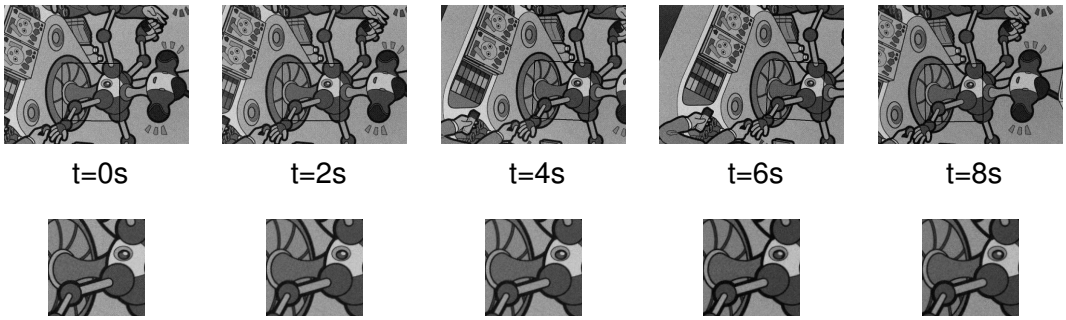

Fig. 3. ESM tracking results on GPU

might induce an even low speed. Besides, Images were extracted from the CPU and GPU ESM tracking sequences and shown seperately in Fig. 2 and Fig. 3. Tracking area is a 200×200 window shown in $t = 0s$. The windows in the first row of Fig. 2 and Fig. 3 are warped back and shown in the second row. Despite illumination changes and image noise, the warped windows should be very close to the reference template when the tracking is accurately performed. During the experiments, we increase the object's moving speed slowly. From the sequences we can see CPU performance becomes poor (for $t = 8s$ the warped area is totally different from the template) when moving is fast while GPU can still performs visual tracking well.

V. CONCLUSIONS

In this paper, CUDA implementations of GPU-ESM is presented in a homography-based visual servo system. By utilizing optimization techniques including memory optimization and memory coalescing, GPU-ESM provides us a better system performance with higher processing speed and reliability. Experimental results validate the efficiency and effectiveness of our CUDA applications. By investigating the optimization techniques adopted in our implementation in detail, this study also makes contribution to the general purpose GPU computation community.

REFERENCES

[1] R. Szeliski, *Handbook of Mathematical Models in Computer Vision*, pp 273-292. Springer, 2006.

[2] S. Benhimane and E. Malis, "Homography-based 2D visual tracking and servoing", *International Journal of Robotic Research*, 26(7): 661.676, 2007.

[3] Shum, H. Y. and Szeliski, R,"Construction of panoramic image mosaics with global and local alignment", *International Journal of Computer Vision*, 16(1): 6384.

[4] S. Benhimane, E. Malis,"Real-time image-based tracking of planes using efficient second-order minimization", IEEE/RSJ International Conference on Intelligent Robots Systems, Sendai, Japan, 2004.

[5] E. Malis, "Improving vision-based control using efficient second-order minimization techniques IEEE International Conference on Robotics and Automation", New Orleans, USA, April 2004.

[6] E. Malis, "An efficient unified approach to direct visual tracking of rigid and deformable surfaces", IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, USA, October 2007.

[7] E. Malis, "Improving vision-based control using efficient second-order minimization techniques", IEEE International Conference on Robotics and Automation, New Orleans, USA, April 2004.