Implementation of Video Abstract Algorithm Based on CUDA

Hui Li(✉), Zhigang Gai, Enxiao Liu, Shousheng Liu, Yingying Gai, Lin Cao, and Heng Li

Institute of Oceanographic Instrumentation, Shandong Academy of Science, Qingdao 266001, China
lihuihuidou@163.com

Abstract. The dynamic video abstract is an important part of video content analysis. Firstly, the objective of motion is analyzed, and the objective of the movement is extracted. Then, the moving trajectory of each target is analyzed, and different targets are spliced into a common background scene, and they are combined in some way. The algorithm uses Gaussian mixture model and particle filter to do a large number of calculations to achieve the background modeling and the detection of moving object. With the increase of image resolution, the computing increased significantly. To improve the real-time performance of the algorithm, a video abstract algorithm based on CUDA is proposed in this paper. Through the data analysis and parallel mining of the algorithm, time-consuming modules of the calculation, such as Histogram equalization, Gaussian mixture model, particle filter, were implemented in GPU by using massively parallel processing threads to improve the efficiency. The experimental results show that the algorithm can improve the calculation speed significantly in NVIDIA Tesla K20 and CUDA7.5.

Keywords: Video abstract · Gaussian mixture model · Particle filter
GPU · CUDA · Parallel computing

1 Introduction

The video abstract is a technique that generalizes the main content of the original video. It is also called video synthesis. With the increasing demand for video data processing and the increasing amount of video data, we need to set up a digest for a long video to quickly browse to make better use of it. By using the video abstract technology, we can not only use words in our content-based video retrieval, but also make full use of audio and video information [1]. Video abstract technology solves the problem that how to present video data effectively and fast access. It uses video content analysis to reduce the video storage, classification and indexing, and improve the efficiency, availability and accessibility of the video. It is the development of video analysis technology that based on content.

The generation of the video abstract uses the Gaussian mixture model and particle filtering to perform a large number of operations to achieve the background modeling and motion target detection tracking. Because of the large computational complexity

X. Gu et al. (Eds.): MLICOM 2017, Part II, LNICST 227, pp. 391–401, 2018.
https://doi.org/10.1007/978-3-319-73447-7_43

and long processing time of Histogram equalization, such as Gauss mixture model and particle filter, it is difficult to apply in real-time video processing with higher real-time requirements [2]. Therefore, the processing time of the algorithm must be effectively reduced to meet more real-time applications.

In recent years, GPU (Graphics Processing Unit) has been applied to large-scale parallel computing and floating point calculation, and its multi-thread and multi-core processors are especially suitable for data parallel computing. CUDA architecture, that uses SIMT (single-instruction-thread, multi-thread) model, is a software platform that can be used to implement fine-grained parallelism. Developing CUDA programs becomes more flexible and efficient because of their easy programmability when accelerating image processing in program-level parallelism. This paper presents a video abstract algorithm based on GPU CUDA, which exploits a large number of parallel threads and heterogeneous memory hierarchy of GPU to improve the execution efficiency.

2 Gauss Mixture Model and Particle Filter

2.1 Gauss Mixture Model

Gaussian mixture Model is a classical adaptive background extraction method presented by Stauffer et al. [3]. It is a kind of method based on background modeling, and it constructs each pixel according to the distribution of each pixel in the time domain of the color distribution model in the video, in order to reach the purpose of background modeling. The Gaussian mixture Model is a weighted sum of finite Gauss functions, which can describe the multimodal state of pixels, and it is suitable for accurate modeling of complex backgrounds such as light gradients and tree swaying.

By looking for a random sample spread in state space to approximate the probability density function, replace the integral operation with the sample mean, and obtains the state minimum variance distribution. The core idea is that through the random state particles extracted from the posterior probability to express the distribution. It is a kind of sequential importance sampling method. It is often used background subtraction method to extract the moving object when the camera fixed. Subtract the current image from background image that previously obtained, if the pixel exceeds a certain threshold, the pixel is identified as the target region; otherwise that is the background area.

The estimation algorithm of single Gauss background model is suitable for indoor environment and outdoor environment which is not very complicated.

The first step is initializing the background image. The average gray value μ_0 of each pixel and the variance σ_0^2 of the pixel gradation in the video sequence image $f(x, y)$ are calculated for a period of time. The initial background image of B_0 with Gauss distribution is composed of μ_0 and σ_0^2. As shown in formulas (1) (2).

$$\mu_0(x, y) = \frac{1}{T}\sum_{i=0}^{T=1} f_i(x, y) \tag{1}$$

$$\sigma_0^2(x,y) = \frac{1}{T}\sum_{i=0}^{T=1}[f_i(x,y) - \mu_0(x,y)]^2 \tag{2}$$

The second step is to update the background image. If the scene changes, the background model needs to respond to these changes. The algorithm updates the background model by using the real-time information provided by the video sequence, where $F_t(x,y)$ represents the real-time image at time t, $B_{t-1}(x,y)$ represents the background image at time t − 1, as shown in formula (3).

$$B_t(x,y) = (1-\rho)\cdot B_{t-1}(x,y) + \rho \cdot F_t(x,y) \tag{3}$$

The background update rate ρ is a constant that reflects the update speed of the current image to the background. Since the influence of the moving target on the background is not taken into account, the pixel on the moving target is also involved in the updating of the background image, resulting in an error in the updated background and the actual background. Thus, Koller et al. improved the algorithm by updating pixels that were labeled as background areas. As shown in formula (4), respectively indicate that $B_t(x,y)$ is judged as background or foreground.

$$B_t(x,y) = \begin{cases} (1-\rho)\cdot B_{t-1}(x,y) + \rho \cdot F_t(x,y) \\ B_{t-1}(x,y) \end{cases} \tag{4}$$

In video surveillance systems, surveillance cameras are generally fixed. If the background is completely stationary, each pixel in the background image can be described by a Gauss model [4]. But in reality, the background is not absolute static, such as the branches swing, or a pixel in the background image at a certain moment may be the sky, may be leaves, may also be branches; each state of the color value of the pixel is different. Therefore, a Gauss model can not reflect the actual background. So, the Gauss mixture distribution is used to describe the background model. The Gauss distribution, which is used to describe the color of each pixel, is K, and K is determined by the computational power and the available memory of the computer, and it generally takes between three and five. The distribution of the currently observed pixel values is as formula (5) shows.

$$P(X_i) = \sum_{i=1}^{k} \omega_{i,t} * \eta\left(X_i, \mu_{i,t}, \Sigma_{i,t}\right) \tag{5}$$

$\omega_{i,t}$ is an estimate of the weight of a Gaussian model at time t, $\mu_{i,t}$ is the mean of a Gaussian model at time t, and $\Sigma_{i,t}$ is the covariance matrix of a Gaussian model at time t. η is a Gaussian probability density function as shown in formula (6).

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t-\mu_t)^{\mathrm{T}}\Sigma^{-1}(X_t-\mu_t)} \tag{6}$$

The covariance matrix is assumed to be as follows.

$$\Sigma_{k,t} = \sigma_k^2 I \tag{7}$$

At time t, the weight $\omega_{k,t}$ of the K distribution is updated as follows.

$$\omega_{k,t} = (1-\alpha)\omega_{k,t-1} + \alpha\left(\mathrm{M}_{k,t}\right) \tag{8}$$

In the formula, α is the learning rate, the matching model $M_{k,t}$ is 1, and the remaining mismatch model is 0. For the mismatched model, the model parameters are unchanged, and the matching parameters in the matched model are updated as follows.

$$\mu_t = (1-\rho)\mu_{t-1} + \rho X_t \tag{9}$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^{\mathrm{T}}(X_t - \mu_t) \tag{10}$$

$$\rho = \alpha\eta(X_t|\mu_k, \sigma_k) \tag{11}$$

Each Gaussian distribution is arranged in order of priority, the former B as the background model, and B is defined as follows.

$$\mathrm{B} = \arg\min_b \left(\sum_{k=1}^{b} \omega_k > \mathrm{T} \right) \tag{12}$$

T is a pre-defined threshold, which can actually reflect the minimum proportion of the data in the background to the total data.

2.2 Particle Filter

In this paper, we use the sequential importance particle filter algorithm, which uses the weighted sum of a series of random samples to represent the required posterior probability density, obtains the estimated value of the state, realizes the tracking of the moving target and obtains the target trajectory [5].

In the Monte Carlo simulation method which based on importance sampling, it is necessary to recalculate the importance weight of the whole state sequence by estimating the posterior filtering probability and using all the observed data. Sequential importance sampling is the basis of particle filtering, which applies the sequential analysis method in statistics to the Monte Carlo method, so as to realize the recursive estimation of the probability density of the posterior filter [6]. Assume that the importance probability density function $q(x_{0:k}|y_{1:k})$ can be decomposed by the following formula.

$$q(x_{0:k}|y_{1:k}) = q(x_{0:k-1}|y_{1:k-1})q(x_k|x_{0:k-1}, y_{1:k}) \tag{13}$$

Set the system state is a Markov process, and individual observations are independent in a given system state, there are,

$$p(x_{0:k}) = p(x_0)\prod_{i=1}^{k} p(x_i|x_{i-1}) \tag{14}$$

$$p(y_{1:k}|x_{1:k}) = \prod_{i=1}^{k} p(y_i|x_i) \tag{15}$$

The recursive form of the posterior probability density function can be expressed as the following formula.

$$\begin{aligned} p(x_{0:k}|Y_k) &= \frac{p(y_k|x_{0:k}, Y_{k-1})p(x_{0:k}|Y_{k-1})}{p(y_k|Y_{k-1})} \\ &= \frac{p(y_k|x_{0:k}, Y_{k-1})p(x_k|x_{0:k-1}, Y_{k-1})p(x_{0:k-1}|Y_{k-1})}{p(y_k|Y_{k-1})} \\ &= \frac{p(y_k|x_k)p(x_k|x_{k-1})p(x_{0:k-1}|Y_{k-1})}{p(y_k|Y_{k-1})} \end{aligned} \tag{16}$$

The recursive form of particle weights $w_k^{(i)}$ can be expressed as formula (17).

$$\begin{aligned} \omega_k^{(i)} \propto \frac{p\left(x_{0:k}^{(i)}|Y_k\right)}{q\left(x_{0:k}^{(i)}|Y_k\right)} &= \frac{p\left(y_k|x_k^{(i)}\right)p\left(x_k^{(i)}|x_{k-1}^{(i)}\right)p\left(x_{0:k-1}^{(i)}|Y_{k-1}\right)}{q\left(x_k^{(i)}|x_{0:k-1}^{(i)}, Y_k\right)q\left(x_{0:k-1}^{(i)}|Y_{k-1}\right)} \\ &= \omega_{k-1}^{(i)}\frac{p\left(y_k|x_k^{(i)}\right)p\left(x_k^{(i)}|x_{k-1}^{(i)}\right)}{q\left(x_k^{(i)}|x_{0:k-1}^{(i)}, Y_k\right)} \end{aligned} \tag{17}$$

In general, it is necessary to normalize the weight of the particle, as formula (18).

$$\varpi_k^{(i)} = \frac{\omega_k^{(i)}}{\sum_{i=1}^{N}\omega_k^{(i)}} \tag{18}$$

The sequential importance sampling algorithm generates the sampled particles from the importance probability density function, and obtains the corresponding weights with the arrival of the measured values. Finally, the posterior filtering probability density is described in the form of particle weighting sum, and then get the state estimate.

3 CUDA Architecture

CUDA (Compute Unified Device Architecture) is a general parallel computing architecture introduced by NVIDIA [7], and the architecture can dramatically improve computational performance by using GPU.

There are many SMs (Streaming Multiprocessor) in a GPU in the hardware architecture of GPU CUDA, these similar to the CPU core, and a SM is equipped with a number of SPs (Streaming Processor). SP, namely CUDA core, is the basic processing unit of CUDA, and the specific instructions and tasks are handled in the SP. GPU parallel computing, that is, the parallel processing of multiple SP.

GPU threads are organized in a grid and each grid contains a number of thread blocks [8]. The thread is the basic execution unit in CUDA, a number of threads forms a thread block and the thread block can be a one dimensional, two-dimensional or three-dimensional structure. Many threads in the same thread block have the same instruction address, which not only can execute in parallel, but also can realize the communication among the blocks through the shared memory and the barrier. The CUDA memory access model is shown in Fig. 1.

CUDA code applies to both the host processor (CPU), but also applies for the device processor (GPU) [9]. The host processor is responsible for deriving a multi-thread task (CUDA called a kernel program) that runs on a GPU device processor. When using CUDA programming, the program is divided into two parts, host side and device side. Host side is executed on the CPU part, and it is the serial code. Device side is executed on the GPU part, and it is the parallel code. Program in device side is also called "kernel", and the grid is composed of all threads generated by kernel [10]. In CUDA, host and device have different memory spaces. When these tasks have

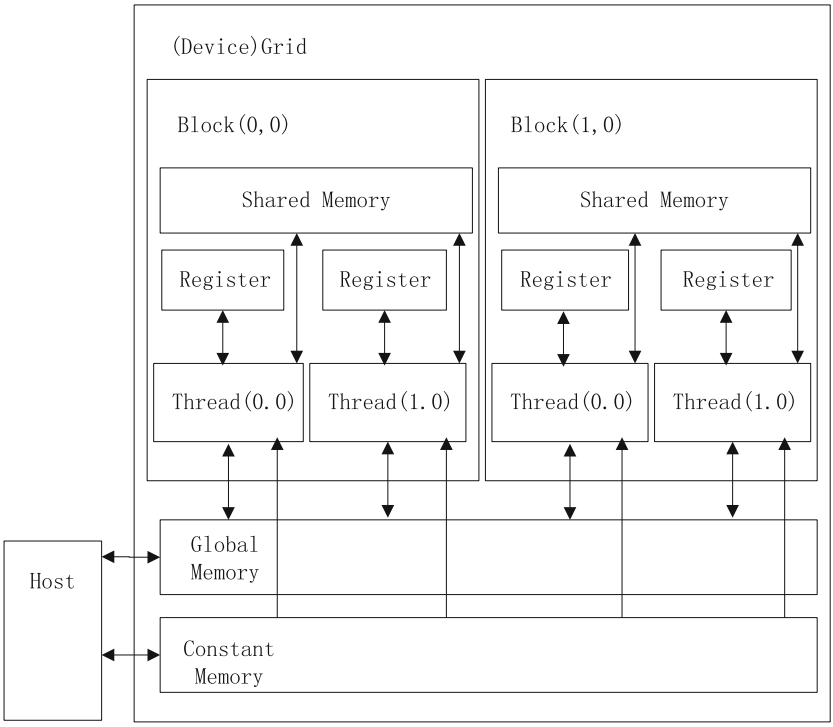


Fig. 1. CUDA memory access model

enough parallelism, with the increase of SM in GPU, the computing speed of the program will be increased. Figure 2 shows the CUDA thread model.

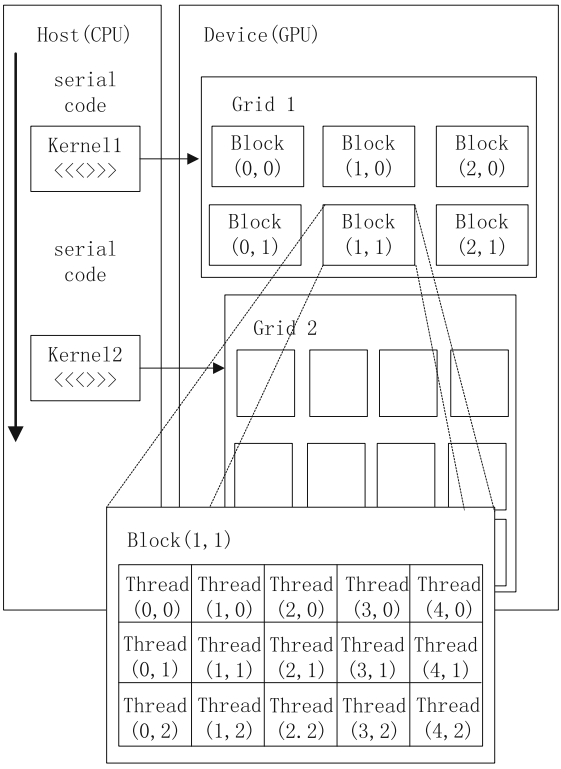


Fig. 2. CUDA program model

4 Realization of Video Abstract Based on CUDA

The R, G and B color components of the image are calculated respectively by the video abstract algorithm when processing the input image. That is, data calculation is independent of each other, so it can take full advantage of SIMT characteristics of CUDA for high-performance parallel processing.

The program based on the CUDA architecture is executed in collaboration in the host side and the device side. In the CUDA programming of MSR algorithm, it should increase the data parallel as much as possible, and reduce the data copy between the host side and the device side, to maximize the advantages of GPU computing. In this paper, the host side is used to realize the reading of the input image, the memory allocation and recovery, and the data transmission between the host and the device. The rest of the implementation process of the algorithm are executed at the device side, mainly Gauss mixture model, erosion, dilation, histogram and particle filter.

In order to achieve higher efficiency of the instruction stream, the algorithm allocates a processing thread for each pixel and uses the shared memory to store convolution operator. Since the size of the CUDA warp is 32, so the number of threads in the thread block is preferably a multiple of 32 to take full advantage of the computing power of each thread.

In this paper, the algorithm is divided according to the size of 64 threads of each thread block according to the two-dimensional allocation method:

dim3 blockSize, gridSize
blockSize.x = 8
blockSize.y = 8.

When the size and dimensions of the thread block are determined, the number of thread blocks in the thread grid can be determined according to the size of the image. In order to avoid the error caused by the processing of the boundary of the image, the determination of the dimensions of the thread grid in the X and Y directions of the algorithm is determined by the following method:

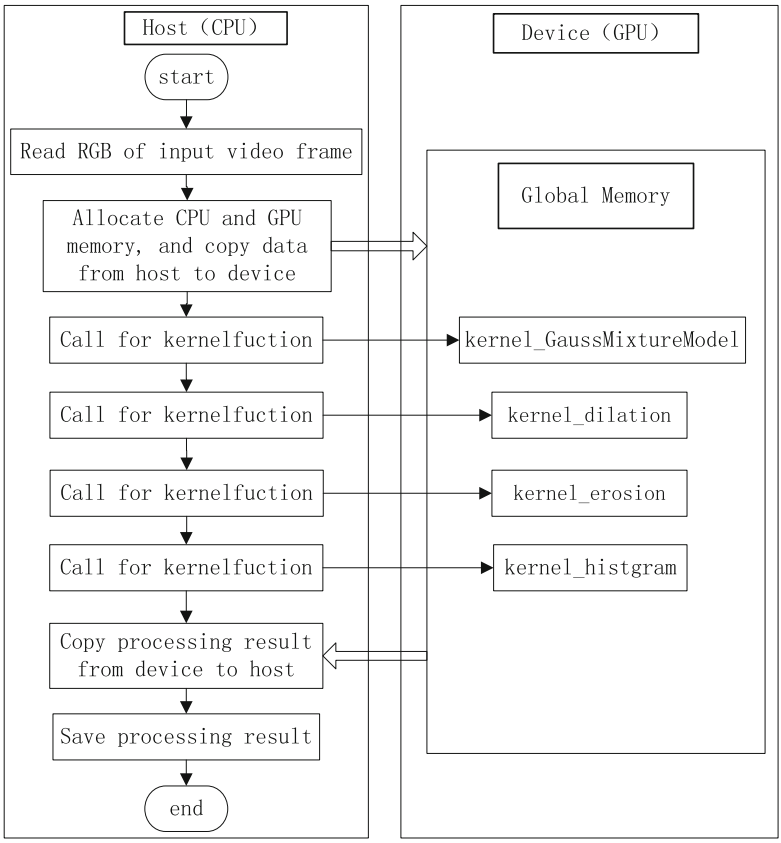


Fig. 3. Flowchart of video abstract algorithm based on CUDA

gridSize.x = (ImageWidth + blockSize.x − 1)/blockSize.x;
gridSize.y = (ImageHeight + blockSize.y − 1)/blockSize.y;

In the case of the histogram equalization, the thread grid in the X, Y, Z direction of the dimension is determined by the following method,

gridSize.x = (Hx[0] * 2 * 2 + blockSize.x − 1)/blockSize.x
gridSize.y = (Hy[0] * 2 * 2 + blockSize.y − 1)/blockSize.y
gridSize.z = N.

Hx and Hy are the size of the filter window, and N is the number of particles. The execution process of the whole algorithm is shown in Fig. 3.

5 Experimental Simulations and Analysis

This algorithm uses NVIDIA Tesla K20 and CUDA7.5 for performance testing, in which the CPU of 2.66 GHz, 2.67 GHz Core Duo. The NVIDIA Tesla K20 has 2496 SPs, and the processing power of single precision is 3524GFLOPS. In this experiment, video frame image with a resolution of 640 × 360 was selected for performance testing. Finally, the speed of the algorithm in CPU and GPU are compared.

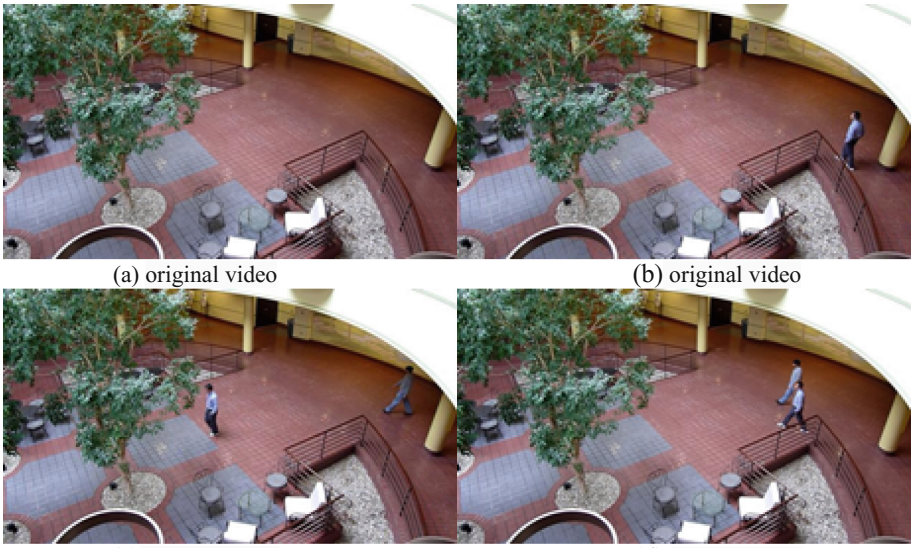
In the CPU algorithm, the basic time that detects a new target of a frame by the first is 180 ms, the filtering time is n times of 15 ms, and the n is the number of filtering. Background modeling frames and tracking multiple frames of detected targets are about 60 ms and 100 ms. For a total of 60 frames, for example, the average frame time is 350 ms, and the core processing time is 150 ms.

In the CUDA algorithm, the basic time that detects a new target of a frame by the first is 80 ms, the filtering time is n times of 15 ms, and the n is the number of filtering. The marker area pixel mu is significantly optimized, the time drops from 25 ms to 1 ms, and the time of the target contour rectangle is extracted from 90 ms to 7 ms. Background modeling frames and tracking multiple frames of detected targets are stabilized to 40 ms and 80 ms. Take a total of 60 frames, for example, the average frame time is 300 ms, and the core processing time is 100 ms.

In the CPU algorithm, statistical histogram algorithm need N dynamic particle multiple iterations. Each iteration takes 20 ms, and each particle filter needs multiple iterations, so it is an obvious bottleneck. After porting the algorithm to GPU, the iteration time fell from 20 ms to 5 ms, and the performance increased by four times.

Figure 4(a) is the background image, Fig. 4(b)(c) is two different goals at the different time in the same scene, Fig. 4(d) sets the two goals in the same frame to form a video abstract.

The experimental results show that the total calculation speed of the video abstract algorithm based on CUDA proposed in this paper is obviously improved compared with the CPU implementation. And the result of the target detection is relatively accurate, the target can be tracked continuously, the formation of the abstract video can effectively save the original video information.



(a) original video (b) original video

(c) original video (d) video abstract

Fig. 4. Video abstract result

6 Conclusions

The significant characteristic of video abstract is browsing all moving targets in the video for several hours in a few minutes. Browse video abstract can greatly shorten the time to view the original video, so it can improve the efficiency and accuracy of artificial recognition. However, the complexity of computing has seriously affected in its actual applicability, especially for high-definition video. Based on the CUDA architecture, this paper proposes a video abstract algorithm based on CUDA. The experimental results show that the processing speed is significantly higher than that of the CPU algorithm, and the real-time performance of the algorithm is better.

Acknowledgments. This work was supported by the Natural Science Foundation of Shandong Province, Grant No. ZR2015YL020.

References

1. Wang, J., Jiang, X., Sun, T.: Summary of video abstract technology. J. Image Graph. **19**(12), 1685–1695 (2014)
2. Tian, H., Ding, S., Yu, C., Zhou, L.: Research on video abstract technology based on target detection and tracking. Comput. Sci. **43**(11), 297–312 (2016)
3. Hua, Y., Liu, W.: Improved Gauss mixture model for moving target detection. J. Comput. Appl. **34**(2), 580–584 (2014)
4. Li, B., Yang, G.: Adaptive foreground extraction of Gauss mixture model. J. Image Graph. **18**(12), 1620–1627 (2013)

5. Li, T., Fan, H., Sun, S.: Particle filter theory and method and its application in multi-target tracking. Acta Autom. Sin. **41**(12), 1981–2002 (2015)
6. Wang, F., Lu, M., Zhao, Q.: Particle filter algorithm. Chin. J. Comput. **37**(8), 1679–1694 (2014)
7. CUDA parallel computing platform [EB/OL]. http://www.nvidia.cn/object/cuda-cn.html
8. Cook, S.: CUDA parallel programming: guide for GPU programming. In: Su, T., Li, D. (eds.) Translated Version.1, pp. 191–200. Mechanical Industry Press, Beijing (2014)
9. Jian, L., Wang, C., Liu, Y., et al.: Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA). J. Supercomput. **64**(3), 942–967 (2013)
10. Yang, N.Z., Zhu, Y., Pu, Y.: Parallel image processing based on CUDA. In: 2008 International Conference on Computer Science and Software Engineering, ICCSSE 2008. IEEE Computer Society, California, pp. 198–201 (2008)