3D Objects Tracking by GPGPU-Enhanced Particle Filter Algorithms

Jieyun ZHOU† Masayuki NUMAO‡ Xiaofeng LI† and Haitao CHEN†

† The School of Communication and Information, University of Electronic Science and Technology of China
No.2006, Xiyuan Ave, West Hi-Tech Zone, 611731, Chengdu, Sichuan, P.R.China
‡ Graduate School of Informatics and Engineering, University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182-0021 Japan
E-mail: † zidongyi@gmail.com, ‡ numao@cs.uec.ac.jp

Abstract Objects tracking methods have been wildly used in the field of video surveillance, motion monitoring, robotics and so on. Particle filter is one of the promising methods, but it is difficult to apply for real time objects tracking because of its high computation cost. In order to reduce the processing cost without sacrificing the tracking quality, this paper proposes a new method for real-time 3D objects tracking, using GPGPU-enhanced particle filter algorithms.

Keyword Objects Tracking, Particle filter, 3D, GPGPU

1. Introduction

In recent years, visual tracking research is getting popular among researchers in the fields of video surveillance, robot vision, etc. A large amount of research papers focus on the improvement of efficiency and accuracy of tracking algorithms. In our approach, tracking problem can be solved by the three basic theories: filtering theory, similarity function, and partial differential equation.

The first way is based on the filtering theory. Objects tracking problem can be converted to the probability density function estimation problem. In this way we often use the Kalman filter[1] or particle filter[2] for tracking. The merits of this method are that we can easily deal with the occluded objects problem. But the drawbacks of this method is that computing complexity is very high so it will take a large amount of calculation time and make real-time performance difficult. So nowadays a lot of researchers concentrate on how to decrease the computing complexity. Second way is based on the Mean-Shift objects tracking[3]. It is based on the probability similarity function between the target model and the target candidate to calculate Mean-shift iteration equation by gradient descent algorithm. The advantage of it is that the computing complexity is not very high. But the shortcoming of this method is that it cannot deal with the occluded objects problem. The last method is based on partial differential equation. Changing the target tracking problem into functional optimization problem, by the functional extremum from a partial differential equation.

We choose particle filter as our tracking method because its ability of solving non-linear and non-Gaussian dynamic system. And it becomes more attractive due to its high robustness and effectiveness. 3D tracking becomes popular nowadays due to the effectiveness[4]. The object itself is a 3D object. But in traditional 2D tracking we directly take the projection of object. So we lose a lot of effective information. By 3D objects tracking we can retrieve these effective information to get high robustness and effectiveness.

General Purpose Graphics Processing Units (GPGPU)[5] recently becomes more and more popular due to its high parallel computing ability and lower cost. Compute Unified Device Architecture (CUDA), utilizes the blocks and the threads of graphics processing units (GPU) to realize the parallel computing. There are some designs of particle filter on GPU. But which part of the algorithm can be migrated into GPGPU and how to parallelize the algorithm to realize the time reduction is still a challenge.

This paper proposes a development of a robust and effective system for object tracking. The design is as follows. First, we use a Kinect to get the 3D information of objects. Unlike the traditional 2D-based objects tracking, 3D objects tracking add depth information. It can track not only from the x and y axis but also from the z axis, and the depth information can correct some errors caused by self-adaptive windows in 2D objects tracking. Second, to solve the high computation cost problem, we decided to use the GPGPU to parallelize the particle filter algorithm. We propose a several way to implement particle filter algorithms on GPU and evaluate the performance by actually running a program using CUDA. We use GPU in the following two area, We use the parallel computing ability of GPU to get a large number of transition particles. We can also calculate the similarity function and the weight of each particles from the histogram by each particles' region using GPU. We can also use optimizing parallel reduction in CUDA method to reduce the computing complexity when calculate the weight of each particles. From the experiment result this tracking method can be more real-time and robust.

This paper is organized as follows. Section 2 describes the background, particle filter and the platform CUDA. Section 3, the two proposed design techniques are demonstrated. The experiment result is showed in section 4. And the section 5 draws the conclusion.

2. Background

2.1 Particle filter

Particle filter is a tracking method based on Monte Carlo sampling. Two kinds of state variables are defined in this method, state variable and measurement variable. In hidden Markov model we can't obverse the state of target very clearly, and only measurement variable is available. From the measurement model we can conjecture the state of state model.

The definition of the state model has been given in [6], the state sequence $\{x_k, k \in N\}$ of a target given by

$$x_k = f_k(x_{k-1}, v_{k-1}) \quad (1)$$

Where f_k is the nonlinear system function, x_k is the state vector, v_k is the noise of this system. k is the index of time step. In tracking problem, state model means actual movement of the target object.

The measurement model can be measured from the state model

$$z_k = h_k(x_k, n_k) \quad (2)$$

Where z_k is the measurement vector, n_k is the measurement noise caused by measure method. h_k is the measurement function. In tracking problem, measurement model means the estimated movement of target candidate. The possibility of target candidates are calculated by the likelihood function.

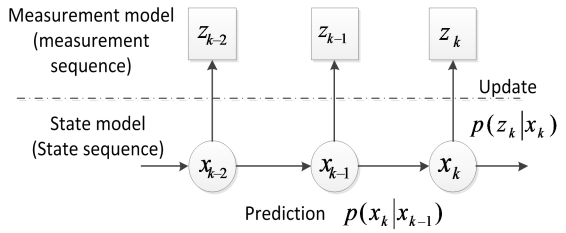


Fig1 state transition graph

Such a filter consists of two stages: prediction and update.

- The prediction stage uses the known system model $p(x_k|x_{k-1})$ to predict the state probability density function (pdf) forward from one measurement time to the next. $p(x_k|x_{k-1})$ defines how the particles moves, it is a dynamic model.

$$p(x_k \mid z_{1:k-1}) = \int p(x_k \mid x_{k-1}) p(x_{k-1} \mid z_{1:k-1}) d_{x_{k-1}} \quad (3)$$

- The update operation uses the latest measurement to modify the prediction pdf. This is achieved by using Bayes theorem, which is the mechanism for updating knowledge about the target state in the light of extra information from new data.

$$p(x_k \mid z_{1:k}) = \frac{p(z_k \mid x_k) p(x_k \mid z_{1:k-1})}{p(z_k \mid z_{1:k-1})} \quad (4)$$

Where the part below is the normalizing constant

$$p(z_k \mid z_{1:k-1}) = \int p(z_k \mid x_k) p(x_k \mid z_{1:k-1}) d_{x_k} \quad (5)$$

Where $p(z_k|x_k)$ is the likelihood function. It defines the similarity of measurement model and state model. The similarity between target model and target candidate is also defined by $p(z_k|x_k)$. So we define the likelihood function as the weights of particles.

$$w_k = p(z_k|x_k) \quad (6)$$

So from the equation (3) (4) (6)and the known system model $p(x_k|x_{k-1})$ we can easily get the state sequence or the target model, fig1 shows how the transition works.

The tracking method based on particle filter can be used as follow.

Algorithm 1 Basic of Particle filter

1: ***for*** i = 1 to N
2: Initial Particles x_k^i
3: ***end for***
3: ***for*** very frame ***do***
4: for i = 1 to N
5: $x_{k\_pri}^i = f(x_{k-1\_post}^i, v_k^i)$ function (1)
6: Weight $w_k^i = p(y_k|x_{k\_pri}^i)$ function (6)
7: Normalize weight.
8: Resampling
9: ***end for***
10: **end for**

2.2 GPGPU calculate by Compute Unified Device Architecture (CUDA)

General-purpose computing on graphics processing units (GPGPU)[7] which use the graphics processing units (GPU) to calculate the data for CPU. GPU is usually used in image processing, but how to migrated it from CPU to GPU is still a challenge.

CUDA is a parallel computing platform and program model invented by NVIDIA. It enables a large increase of computing performance by migrating the data from CPU to GPU by using parallel computing ability of GPU. A new execution model has been used is Single instruction multiple thread (SIMT). SIMT use 32 parallel threads to create, manage and execute. We call it *warp*. A *warp* executes one common instruction at a time, so full efficiency is realized when all 32 threads of a *warp* agree on their execution path.

Data-parallel processing maps data elements to parallel processing threads. Many applications that process large data sets can use a data-parallel programming model to speed up the computations. In 3D rendering, large sets of

pixels and vectors are mapped to parallel threads. Similarly, image and media processing applications such as post-processing of rendered images, video encoding and decoding, image scaling, stereo vision, and pattern recognition can map image blocks and pixels to parallel processing threads. In fact, many non-image-processing algorithms can be accelerated by data-parallel processing, from general signal processing or physics simulation to computational finance or computational biology.

2.3 Kinect for 3D information

Kinect is a motion sensing input device by Microsoft for the Xbox 360 video game console and Windows PCs[8]. Based around a webcam-style add-on peripheral for the Xbox 360 console, it enables users to control and interact with the Xbox 360 without the need to touch a game controller, through a natural user interface using gestures and spoken commands.

The device features, RGB camera, Depth sensor (IR), Multi-array microphone, Motor to adjust camera angle. It can get three kinds of informations, Color image, Depth image and Skeleton.

We use a Kinect as a camera for objects tracking. We use RGB information for normal tracking. Depth information from Kinect is used to correct the tracking area.

3. Design method for 3D objects tracking improvement

Because of the time consuming algorithm, it is a critical issue for tracking object in a real time. In this section, two design techniques have been proposed , (A) 3D self-adaptive tracking window. (B) Parallelization on the transition part and the likelihood part.

In the following explanations we will use the following notations,

(1) Tracking window. The target object has been chosen in the first frame, and the surrounding rectangle is defined as tracking window. Some of image pixels are sampled as particles.

(2) Particle. The basic unit of the particle filter algorithm. It moves in every frame to confirm the location of the tracking window.

(3) Block. The basic unit in GPU. For parallel computation.

(4) Thread. The basic unit in GPU. For serial computation.

3.1 3D self-adaptive tracking window

Traditional particle filter tracking method uses the random number generator to get random sizes of the tracking windows[8]. It uses N random numbers to get N random scales and compute similarity function values. The window which has the maximum value should be the target window. This method may have some shortcomings. Even if the point is in the target window point, the similarity function value can become very small because the window size is different. So we need another way to calculate the correct size of tracking window. We propose a method of 3D self-adaptive tracking window. Scale is used for describing the changes of tracking window.

If we use the depth information, we can get the correct value of the tracking window by the first chosen window and the distance between camera and target. So it doesn't need N random numbers.

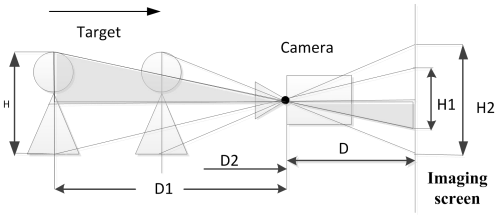


Fig2 Relation between distance and size of window

From the triangle similarity function we can get the equation of the gray part as below.

$$\begin{cases} \dfrac{H/2}{D_1} = \dfrac{H_1/2}{D} \\ \dfrac{H/2}{D_2} = \dfrac{H_2/2}{D} \end{cases} \Rightarrow \; D_1H_1 = D_2H_2 \qquad (7)$$

In equation (7) we can get D_1 and D_2 from Kinect by distance sensor, and H_1 by the first selected tracking window. So from equation (7), we can get the size of each particle. Compared with random window size, the size calculated by Kinect is much more correct than before.

The difference between the original version and proposed version is to "get particle" algorithm. It is shown as below.

Algorithm 1 Basic of Get particles	Algorithm2 improve of Get particles
1: ***for*** i = 1 to N	1: ***for*** i = 1 to N
2: Get random number : x,y,s;	2: Get random number : x,y;
3: Sample new particles ;	3: Sample new particles ;
4: Use second-order autoregressive dynamics	4: Use second-order autoregressive dynamics
5: Get new point (xp,yp);	5: Get new point (xp,yp);
6: Get new size of window (sp);	**6: From new point (xp,yp) get depth (dp);**
7: Compare Similarity function;	**7: Calculate size of window (sp);**
8: Find the largest;	8: Compare Similarity function;
9: ***end for***	9: Find the largest;
	10: ***end for***

3.2 Parallelization in particle filter

If we do every step of algorithm1 in CPU it's very slow because it must repeat the same kind of calculation for N times. If we can do all the calculation in one time it can reduce the time cost.We will use GPGPU to realize the parallel computation[9][10]. Fig.3 shows the basic design on GPU.

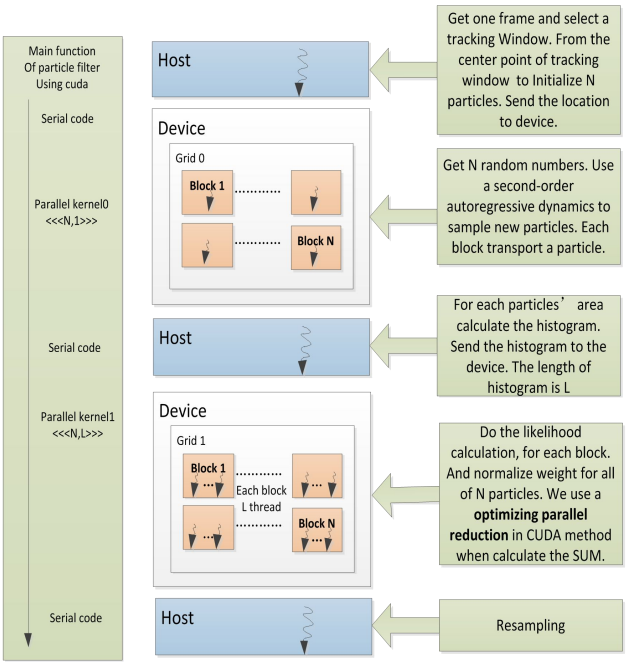


Fig3 Block design of proposed particle filter on CUDA

- **Initialize N particles (CPU).** This part is done in CPU. We first have to choose a tracking window. And then produce N pairs of (x, y) random numbers. This random number set can be moved to GPU and stored in each block. The present location(x0, y0) is also sent to each block.
- **Transport of N particles (GPU).** In this step, in each block that we use random numbers to compute second-order autoregressive dynamics to sample new particles, which get a new position of each particle (xp, yp). In GPU the data is stored in N blocks. Each block is independent and can calculate parallelly in one time. Then new particles' position is sent back from GPU to CPU.
- **Histogram calculation (CPU).** For each particle's position, the distance information is added by Kinect, the size of the area of each particle is calculated on GPU. From the area we can get the histogram of each target candidate. Send N area's histogram (The length of each histogram is L) to N blocks and each block exploit L threads on GPU, each histogram array put in one shared thread.
- **Likelihood calculation (GPU).** In each block likelihood between the histogram of each particle's area and the previous target histogram is calculated. Then the likelihood should be normalized. In order to calculate the average, we need to add all the blocks value together. In this situation we can use an optimizing parallel reduction in CUDA method to optimizing the processing speed. When the calculation results have been obtained, we take the weight data from GPU to CPU again to continue the next calculation.
- **Resampling.** Processed on CPU to produce more particles on larger weight area, on the other hand remove the particles which have small value of weight.

The designed method's pseudocode is showed as follows.

Algorithm 3 Particle filter designed on GPU

1: ***for*** i = 1 to N
2: Initial Particles x_k^i
3: ***end for***
3: ***for*** very frame ***do***
4: move data from CPU to GPU
5: on each block do $x_{k\_pri}^i = f(x_{k-1\_post}^i, v_k^i)$
6: move particles from GPU to CPU
7: ***for*** each particle ***do***
8: calculate **histogram**
9: ***end for***
10: move data from CPU to GPU
11: on each block calculate weight $w_k^i = p(y_k | x_{k\_pri}^i)$
12: Calculate SUM. **optimizing parallel reduction**
13: Normalize weight.
14: move particles from GPU to CPU
15: Resampling
16: ***end for***

- **Parallel Reduction.** The normalization processing needs to add a lot of summation of large numbers. Optimizing parallel reduction is the method how to make best use of the parallel ability which showed in Fig.4. So it reduces the calculation time for getting sum from O(N) to O(log(N)).

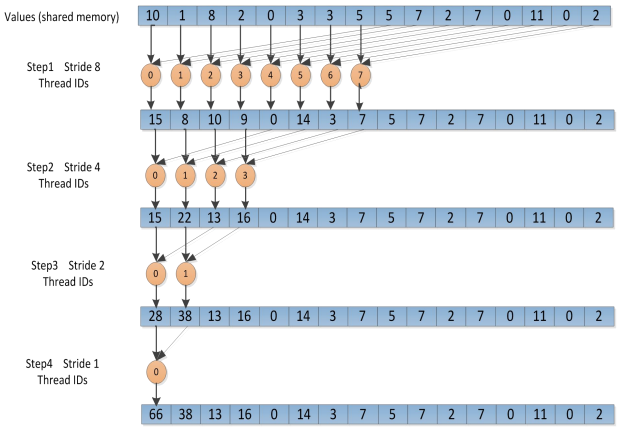


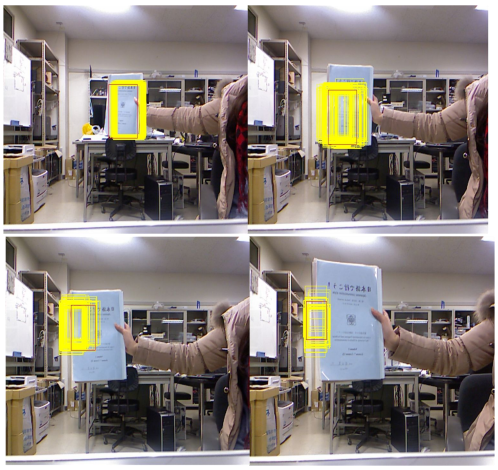
Fig4 Way of calculate sum in parallel

4. Experiment Result

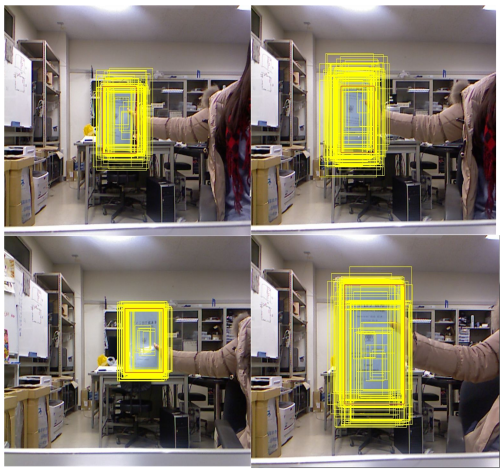
The experiment environment is Windows 7 and Visual Studio 2012 combined with OpenCV 2.3.1, gsl 1.8, and CUDA 5.5. The camera is Kinect 360. The GPU of this experiment platform is NVIDA GeForce 9300M Gs 512M and NVIDA GeForce GTX 480. In this part we compare the different effects and time cost of original algorithm and the proposed algorithm in different number of particles. We can also compare the time cost of each part of particle filter algorithm.

GPU	GPU1	GPU2
Model	nVIDA GeForce 9300M	nVIDIA GeForce GTX 480
GPU memory	256MB	1536MB
MB	64bit	384bit
Memory Clock	1000MHz	3696MHz
Bus bandwidth	8GB/s	177.4GB/s
Main Clock frequency	567MHz	700MHz
RAMDAC frequency	400MHz	400MHz

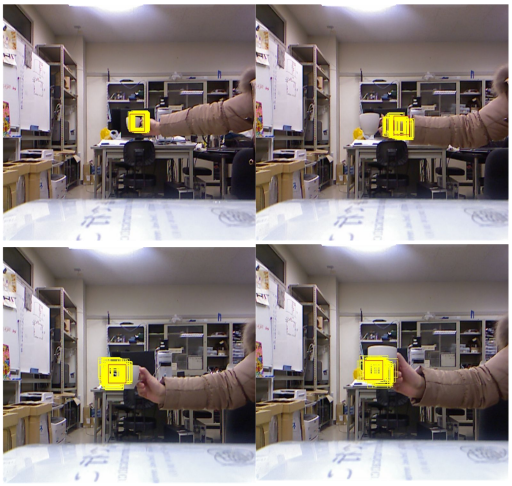
Table 1 Hardware information for evaluation



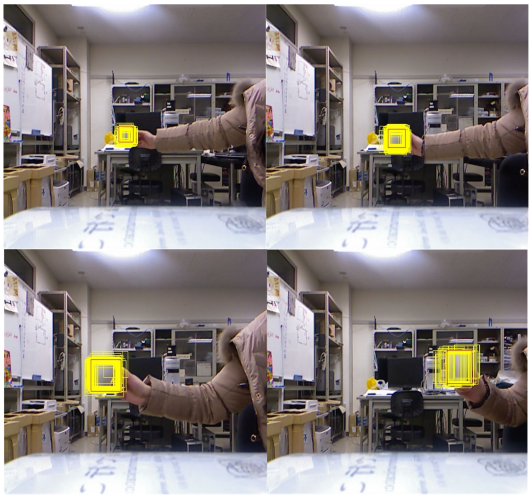
(a)



(b)



(c)



(d)

Fig4 Experiment result by Kinect

In Fig4 (a) is the original algorithm of particle filter, the target is a book. (b) is the new algorithm of 3D tracking by Kinect, the target is also a book. (c) is the original algorithm of particle filter, the target is a small cup. (d) is the new algorithm of 3D tracking by Kinect, the target is also a small cup. These yellow windows show each particles' area, and the red window shows the final location of the target. The original algorithm sometimes loses the target and the tracking window does not fit the target. But the proposed method shows that the target window is self-adaptive by distance.

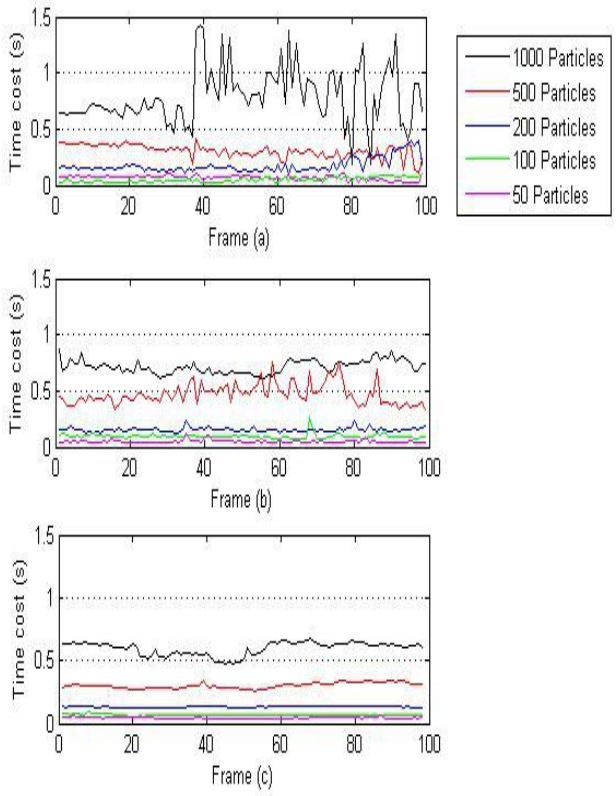


Fig5 Time cost result of new algorithm. (a) is simulated by CPU (b) is simulated by GPU1 (c) is simulated by GPU2

We evaluated the result of the whole time cost in Fig 5. The experiment result has showed that the proposed algorithm has a higher real-time ability and robustness. Then we move to each part. First fig6 shows the transport part of each particle on two different GPU.

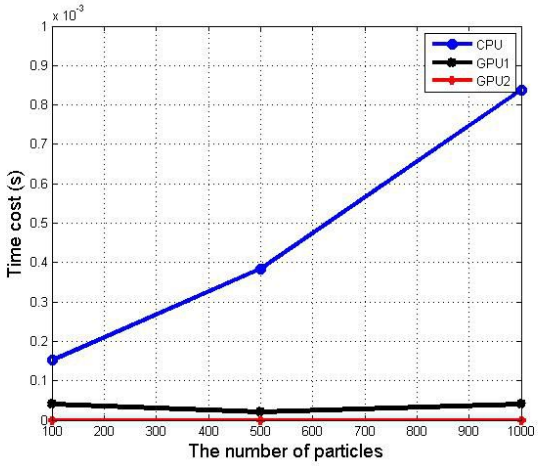


Fig6 Time cost of Transport of N particles (GPU).

Second fig7 shows the likelihood calculating part. We can see the different performance of these two different GPU. On GPU2 we also use the Parallel Reduction method.

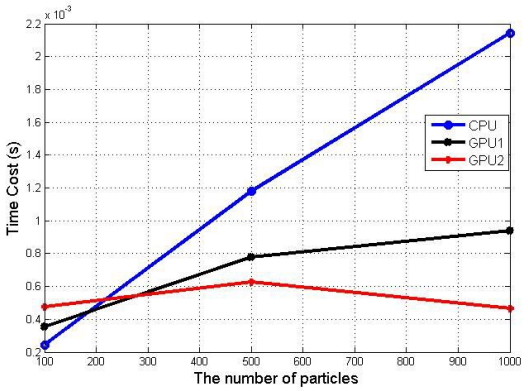


Fig7 Time cost of Likelihood calculation (GPU).

5. Conclusion

From the method we mentioned above we propose two design techniques. (A) 3D self-adaptive tracking window. (B) Parallelization on the transition part and the likelihood part. And from the experiment we have done on CUDA the proposed design can reduce global operation, provide significant speedup and shows a very good robustness.

Reference

[1] R.E.Kalman A new approach to linear filtering and prediction problems. Vol.82D,March 1960,pp. 35-45.

[2] M. SanjeevArulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking of IEEE International Conference on Acoustics 2002, 723-737

[3] D. Comaniciu and P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. 603-619, May 2002.

[4] Iason Oikonomidis ,Efficient Model-based 3D Tracking of Hand Articulations using Kinect, 2012

[5] Nan Zhang, Yun-shan Chen, Jian-li Wang. Image parallel processing based on GPU. In International Conference on Advanced Computer Control(ICACC), 2010. Shenyang:2010.367-371

[6] Min-An Chao, Chun-Yuan Chu, Chih-Hao Chao, and An-Yeu (Andy) Wu EFFICIENT PARALLELIZED PARTICLE FILTER DESIGN ON CUDA. IEEE 2010,299-304

[7] NVIDIA CUDA Programming Introduction 2008 6/7

[8] Microsoft Corp. Redmond WA. Kinect for Xbox 360.

[9] Changhyun Choi, Henrik I. Christensen, "RGB-D Object Tracking: A Particle Filter Approach on GPU," in Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013.

[10] Iason Oikonomidis ,Efficient Model-based 3D Tracking of Hand Articulations using Kinect, 2012

[11] Maggio. E, Cavallaro, A. Hybrid Particle Filter and Mean Shift tracker with adaptive transition model. In Proceedings of IEEE International Conference on Acoustics,Speech, and Signal Processing(ICASSP 2005): 221-224