View Priority Based Threads Allocation and Binary Search Oriented Reweight for GPU Accelerated Real-Time 3D Ball Tracking

Yilin HOU†(a), Ziwei DENG†, Xina CHENG†, Nonmembers, and Takeshi IKENAGA†, Member

SUMMARY In real-time 3D ball tracking of sports analysis in computer vision technology, complex algorithms which assure the accuracy could be time-consuming. Particle filter based algorithm has a large potential to accelerate since the algorithm between particles has the chance to be paralleled in heterogeneous CPU-GPU platform. Still, with the target multi-view 3D ball tracking algorithm, challenges exist: 1) serial flowchart for each step in the algorithm; 2) repeated processing for multiple views’ processing; 3) the low degree of parallelism in reweight and resampling steps for sequential processing. On the CPU-GPU platform, this paper proposes the double stream system flow, the view priority based threads allocation, and the binary search oriented reweight. Double stream system flow assigns tasks which there is no data dependency exists into different streams for each frame processing to achieve parallelism in system structure level. View priority based threads allocation manipulates threads in multi-view observation task. Threads number is view number multiplied by particles number, and with view priority assigning, which could help both memory accessing and computing achieving parallelism. Binary search oriented reweight reduces the time complexity by avoiding to generate cumulative distribution function and uses an unordered array to implement a binary search. The experiment is based on videos which record the final game of an official volleyball match (2014 Inter-High School Games of Men’s Volleyball held in Tokyo Metropolitan Gymnasium in Aug. 2014) and the test sequences are taken by multiple-view system which is made of 4 cameras locating at the four corners of the court. The success rate achieves 99.23% which is the same as target algorithm while the time consumption has been accelerated from 75.1ms/frame in CPU environment to 3.05ms/frame in the proposed system which is 24.62 times speed up, also, it achieves 2.33 times speedup compared with basic GPU implemented work.

key words: real-time, Graphics Processing Unit (GPU), parallelism, thread allocation, particle filter, ball tracking

1. Introduction

In sports analysis field, 3D ball tracking plays a crucial role in obtaining the ball’s position, velocity, and trajectory for TV contents and tactical analysis. However, to meet the high tracking accuracy, the algorithms usually carry a heavy computation complexity, which increases the difficulty in real-time implementation.

Due to the development of computer vision technology and heterogeneous computing method, real-time tracking has shown its importance and value for various utilization. Some works[1], [2] have been focusing on this field. One prevailing tracking algorithm, KLT, tracks the target by feature detection[3]. However, when it comes to 3D ball tracking, the complex circumstances and uncertainties in ball games would cause numerous noises. Another object tracking method well known by people is to use Kalman filter or Kalman extension[4]. Yet it fails to deal with multi-modal stochastic processes which is induced by nonlinear problems. What’s more, when applying the tracking algorithm to realities, work[5] tracks soccer using players tracking result which could cause high computational complexity while work[6] uses only one camera which cannot overcome the occlusion problem. Particle filter algorithm[7], [8], on the other hand, is an attractive option since it is a simulation-based sequential Monte Carlo (SMC) method which could provide state estimation in nonlinear dynamic systems. Also, with multiple synchronized cameras, the occlusion problem can be overcome. Thus, we choose to implement multi-view 3D ball tracking by particle filter[9] to access real-time, which mostly satisfies our demand.

The chosen algorithm[9] of 3D ball tracking by particle filter achieves a highly accurate tracking result from its robust tracking system. Also, the multi-view[10] observation model is applied to solve the occlusion problem. However, the main drawback is the requirement of the computational load. For the application in ball tracking, real-time performance is necessary for its timeliness and quality. As to the chosen algorithm, because of the prohibitively large number of simulations which is also known as particles, the extreme large computation complexity is shown as a problem in achieving real-time performances. An extension work[24] of the chosen algorithm[9] on ping-pong ball tracking and its accelerated work[28] have shown that algorithms basis on particle filter have large potential for acceleration. Basing on the characteristic of ping-pong ball during games, work[24] makes some improvements in the observation and the resampling algorithms of the chosen one[9] which are not friendly to other ball games. What’s more, the modification has caused a considerable distinction in accelerated structure[28] on GPU system which means that it[28] cannot be applied in the acceleration of the chosen algorithm[9]. The goal of this research is implementing a 3D ball tracking system to obtain both high accuracy and real-time realization based on the chosen algorithm[9].

The rise of general-purpose graphics processing unit
Hardware (GPU) makes massive parallelization of algorithms possible. As the rapid development of GPU technology, GPU is prevailing for its significant parallel computing ability in both the graphics processing and the general purpose computing\cite{11, 12}. The independent tasks GPU distributes lie on many computing units that lead to significant acceleration.

Some particle filter based algorithms have been implemented on GPU\cite{13, 14} and achieved significant improvement of speed-up ratio. Yet, as to 3D ball tracking algorithm in multiple view observation model using particle filter\cite{9}, big challenges still exist. 1) The processing flowchart for each frame in the video is sequential as work\cite{23} whereas tasks in particle filter algorithm include data dependency between each other. 2) The multi-view observation model in observation task would cost multiple times computation complexity for one particle. Same likelihood calculation for each particle is needed from each view. Thus, repeated processing would be done for each particle where input is the only difference. 3) The reweight and resampling component is critical to parallelization since it contains most data transmission between particles in the whole system.

To solve the first problem,\cite{17} broke the framework into several queues to achieve the parallelism between the queues. However, only in his application and framework could the assignment of queues be applied. The relation in time flow was not stated clearly, let alone the feature extraction and grid extraction which is not utilized in our algorithm. As to the second problem, work\cite{18}, \cite{19} settled the threads for each task in the algorithm as the same. However, when assigning the multi-view algorithm, repeated work for each thread corresponding to each particle still exists. Lastly, some work\cite{17, 20, 21} has been focusing on this problem in reweight task by applying sum reduction method\cite{26}. Although they have reduced the time complexity by achieving partial parallelism summation for reweight, time complexity was still large.

In this paper, based on 3D ball tracking algorithm using particle filter, double stream system flow, view priority threads allocation and binary search oriented reweight are proposed.

Firstly, view priority threads allocation accelerates the memory accessing in observation task where image data are needed. In the observation task, the threads number are multiplied to stand for each particle’s distinct projected situation in each view. Also, taking views as the priority to assign the threads helps to achieve parallelism in both memory accessing and calculation processing. Then, binary search oriented reweight reduces the computation and time complexity in reweight calculation and selection in resampling. Finally, double stream system flow assigns the tasks into different streams according to the data dependency to lower the serial flowchart executing time.

The paper is arranged as below. Section 2 introduces the structure of the chosen tracking algorithm and the system it builds. Section 3 provides a detailed illustration of the proposed system and the proposals which include three ideas. Followed Sect. 4 discusses the experiment result and considerations while Sect. 5 is the conclusion.

2. Particle Filter Based 3D Ball Tracking Algorithm in Multi-Views

The framework of the chosen tracking algorithm\cite{9} is shown in Fig. 1, which consisted in prediction, projection, observation and resampling parts.

The state vector $\mathbf{X}_k$ at each discrete time $k$ is:

$$\mathbf{X}_k = \left[ \begin{array}{c} \mathbf{S}_k \\ \mathbf{V}_k \end{array} \right], k \in \mathbb{N}^*$$

(1)

where $\mathbf{S}_k$, $\mathbf{V}_k$ and $\mathbb{N}^*$ represent the 3D position of the target, velocity of the target and the set of positive integers.

The time evolution of $\mathbf{X}_k$ is:

$$\begin{pmatrix} \mathbf{S}_k \\ \mathbf{V}_k \end{pmatrix} = \begin{pmatrix} 1 & \Delta T \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{S}_{k-1} \\ \mathbf{V}_{k-1} \end{pmatrix} + \begin{pmatrix} \Delta T^2 \\ \Delta T \end{pmatrix} \mathbf{W}_k,$$

(2)

where $\mathbf{W}_k$, $k \in \mathbb{N}^*$ is the noise term and $\Delta T$ is the sampling time interval.

$$\mathbf{W}_k = A \cdot \omega, \omega \sim \mathcal{N}(0, \begin{bmatrix} \delta^2 & \delta^2 \\ \delta^2 & \delta^2 \end{bmatrix}).$$

(3)

In this equation, $A$ is a constant number and $\omega$ is the system noise which follows Gaussian distribution whose variance is $\begin{bmatrix} \delta^2 & \delta^2 \\ \delta^2 & \delta^2 \end{bmatrix}$.

In observation task, candidates of each particle prediction are observed in multiple views of 2D spaces.

For each view, Eq. (4) shows the likelihood $\mathcal{L}(\mathbf{X}_k; \mathbf{y}_k^m)$ in the $m$th view:

$$\mathcal{L}(\mathbf{X}_k; \mathbf{y}_k^m) = \mathcal{L}_{\text{color}}(\mathbf{X}_k; \mathbf{y}_k^m) \times \mathcal{L}_{\text{foreground}}(\mathbf{X}_k; \mathbf{y}_k^m).$$

(4)

In this equation, $\mathbf{y}_k^m = \{\mathbf{y}_k^1, \mathbf{y}_k^2, \ldots, \mathbf{y}_k^m, \ldots, \mathbf{y}_k^M\}$ is the observation space which is a collection of image frames at discrete time $k$ from total $M$ views where $\mathcal{L}_{\text{color}}$ stands for the color likelihood and $\mathcal{L}_{\text{foreground}}$ stands for the foreground likelihood.

![Fig. 1 Particle filter based 3D ball tracking algorithm in multi-views](image-url)
For color likelihood, it is represented in equation shown below:

\[ L_{\text{color}}(x_k^i; t_k^m) = \sum_{i=0}^{i=N} d_i, \]  

(5)

here \( L_{\text{color}}(x_k^i; t_k^m) \) represents the color likelihood while \( d_i \) is the distance between the \( i^{th} \) template and the observing image region. \( N \) is the total number of the templates.

\[ d_i = \sum_{n=1}^{n=100} (h_{i,n} - H_n)^2. \]  

(6)

Here, \( h_{i,n} \) denotes to the \( n^{th} \) dimension’s value of the \( i^{th} \) template’s histogram and \( H_n \) represents the \( n^{th} \) dimension’s value in the histogram of the particle’s observing image region. The histogram is calculated from pixels value in HSV channel which contains one hundred dimensions.

The other foreground likelihood \( L_{\text{foreground}}(x_k^i; t_k^m) \) is calculated as this formula:

\[ L_{\text{foreground}}(x_k^i; t_k^m) = \sum_{j=0}^{j=M} v_j * I_j / R. \]  

(7)

In this formula, \( I_j \) means the Euclidean distance between the \( j^{th} \) pixel and the tracking center of the observing image region. \( v_j \) means the \( j^{th} \) pixel’s background subtraction value, \( M \) means the amount of pixels in observing image region and \( R \) means the radius of observing image region. The observing image region for likelihood calculation is a circle. The radius \( R \) of it is related to the pixel size of the template and the particle’s position in the 3D space.

With the likelihood model illustrated above, the observation model is as Eq. (8) which represents the likelihood in 3D system model of the \( p^{th} \) particle:

\[ L_p(x_k) = \prod_{m=1}^{M} L(x_k^i; t_k^m). \]  

(8)

The posterior distribution \( p(x_k^i|x_k) \) after state estimation is illustrated as Eq. (9) with \( W_k^p \) stands for the normalized importance weight:

\[ p(x_k^i|x_k) \approx \sum_{p=1}^{P} W_k^p \delta(x_k^i - x_{k-1}). \]  

(9)

Here, \( P \) represents the total number of the particles and \( W_k^p \) follows Eq. (10). \( \delta(x_k^i - x_{k-1}) \) is a binary number which only be 1 when \( x_k^i = x_{k-1} \).

\[ W_k^p = \frac{L_p(x_k^i)}{\sum_{p=1}^{P} L_p(x_k^i)}. \]  

(10)

This work is based on four views’ synchronous videos so that there are four images of 2D spaces in the observation task. Apart from initialization, video input and output, the other parts can be implemented on GPU. As Fig. 2 demonstrated, six tasks are divided and assigned as kernels (K) into GPU and three proposals (P1, P2, P3) are noted: K1. prediction, K2. observation, K3. reweight, K4. resampling, K5. convert-color and K6. random number generator (RNG). Kernels are assigned based on the data transmission between them and the demand of synchronization of threads for distinctive tasks. Kernels are executed inside GPU and the tracking results are transmitted back to CPU. Generally, the parallelism between particles is achieved on GPU in kernels.

3. Proposed GPU Accelerated Real-Time 3D Ball Tracking System

3.1 Structure of CPU-GPU System

The structure of the CPU-GPU heterogeneous platform is as shown in Fig. 2. For each kernel, the independent task is done in GPU following the instruction given by CPU.

In K1. prediction kernel, there are two essential aspects: the random number generator and the space projection. The 3D position of the target \( S_k \) is predicted with Gaussian system noise, so that three Gaussian random numbers are needed for each particle to specify three dimensions. What’s more, a uniform random number is required for each particle in the resampling part, so one more random number is generated here and transmitted to the resampling kernel by using global memory buffer. After that, the 3D positions are projected to 2D space for likelihood evaluation based on camera calibration.

The likelihood is calculated for each particle in K2, the observation kernel. It is based on the image masks generated in K5 which is the CvColor kernel. In K2, the global size (thread count) is enlarged to four times of particles number in order to parallel the likelihood calculation of four views. The color likelihood and the foreground likelihood are calculated based on the data of the masks. In order to reduce the time cost in memory access in these two kernels, vector data type is utilized.

In K3, reweight kernel, the weight of each particle is normalized based on its likelihood, and the cumulative distribution function (CDF) is calculated for particle selection.
in the resampling part. The calculation of CDF is data dependent on each thread so that binary search oriented reweight is proposed to improve the performance which is introduced in Sect. 3.4.

K4, kernel resampling, is to select particles based on binary search and estimate the 3D position and velocity results based on the selected particles’ position and their weight. After finishing this kernel, the 3D position and velocity results are transmitted to CPU for video output.

K5 and K6 are not steps in particle filter but essential data building in chosen tracking system which are related to the first proposal double stream system flow. They are introduced in Sect. 3.2.

### 3.2 Double Stream System Flow

In K5, converting color channel is needed to achieving data of both pixel value in HSV channel and inter-frame background subtraction data. Being the most important preparation work for observation, the result of K5 is necessary for K2 (observation kernel) and relates to K2 only. In K6, one group of uniform random numbers is generated for selecting particles and three groups of Gaussian random numbers are generated for particles position noises in each axis. The Gaussian random numbers are transported to K1 and the ordinary random numbers are given to K5. While there is no data dependency between K6 and K2, K3, two streams are declared to access the parallelism between tasks.

The detail is stated in Fig. 3. Compared with the single flow stream structure which contains all tasks in one stream, the first stream involves K1, K2, K3 and K4 and the other one includes K5 and K6. In GPU implementation, these two streams operate simultaneously. K5 starts after having images information achieved in the current frame and before K2 starts. Thus, execution time overlaps only with K1 in stream1. For K6, the uniform random number it generates is given to K4 in the current frame. The Gaussian random numbers, on the other hand, are utilized in K1 for the next frame. What’s more, the data K6 creates is saved in global buffers and must not be overwritten before K1 finishing reading. In this way, K6 starts right after K1, finishes before K4 starting and executes at the same time with K2 and K3. In that case, K5 and K6 are able to be assigned in one stream with distinctive time overlapped kernels and have no time constraint between each other.

### 3.3 View Priority Based Threads Allocation

View priority based threads allocation is applied only in K2. In this task, every particle needs to calculate their likelihoods in 2D space in four camera views. The 2D position of the particles in each camera view are obtained in K1 with calibration matrix from 3D to 2D.

As illustrated in Sect. 2, $L_p(\mathbf{x}_k)$ is the likelihood in 3D system model of the $p$th particle while $L_{p,m}$ is the image likelihood in $m$th view of the $p$th particle.

In the observation task, multiple views’ data are transported into global memory of GPU for observation. With four cameras’ scene, the definition for the threads number is required to be four times of particles’ number. While one particle has four threads, each thread corresponds to one camera view’s likelihood calculation. During the processing of observation, the calculation for each thread accesses the image data. Since these data are stored in the different addresses, tactical threads allocation should be considered for the good of the memory accessing and algorithm processing.

Input images which save in the global memory are in array form. And the threads which stand for different particles in distinctive views could access them in the observation processing. Generally, the allocation of threads would be assigned as particle priority, which means that the neighbor threads are assigned to process different views of one particle. In this way, developers could locate the index of threads and corresponding particle or view easily. However, for distinctive views’ data, distance between the address indexes is considerable, the memory access processing comes sequential and time-consuming.

In our proposal, we allocate the threads with view priority as shown in Fig. 4. In this method, comparing to particle priority, threads in neighbors would be assigned as diff-
ferent particles in the same view. When likelihood is calculated in the same view, the corresponded threads access same memory with neighbor index, and the memory accessing is coalesced. As [22] introduced, broadcasting appears between neighbor threads in one work group. It is considered that the memory accessing is paralleled in K2.

3.4 Binary Search Oriented Reweight

Reweight and resampling kernels include all data dependency in the chosen algorithm. The core of most standard schemes is a cumulative sum of weights, which is also called prefix sum. Cumulative distribution function (CDF) has apprehensible meaning, as the result of the cumulative ordered sum, it can be used for the general binary search. In our proposal, binary search oriented reweight decreases time consumption by avoiding generating CDF. And binary search with transformed selected number is achieved.

After observation, each particle gets its likelihood in the current frame. Their weights are normalized as Eq. (10).

In general work, summations of particles’ weight according to Eq. (11) composed a CDF from 0 to 1:

\[ C_k^p = \frac{\sum_{i=1}^{p} (W_k^i)}{\sum_{i=1}^{P} (W_k^i)}. \]  

(11)

It should be informed that \( W_k^i \) stands for the weight of particle \( i \). \( C_k^p \) is the cumulated weight for particle \( p \) with the total number \( P \). This function is used for selection in the resampling task. The selected candidate \( s \) satisfies the condition that \( C_k^{s-1} < u < C_k^{s} \). A uniform random number \( u \) between 0 and 1 is required here. The probability for particle \( i \) being selected should be proportional to \( C_k^{i} - C_k^{i-1} \). And the uniform random number can eliminate the influence of particles’ summing up order. Since the process is an iteration of \( \sum_{i=1}^{p} (W_k^i) = \sum_{i=1}^{p-1} (W_k^i) + W_k^p \), the computing complexity of the CDF building processing has always been \( O(N) \). Some work [17], [20], [21] proposed methods of sum reduction [26] in reweight and resampling, however, they only reduced time consumption in achieving \( \sum_{i=0}^{P} (W_k^i) \).

As Fig. 5 shows, the proposed reweight method seems to be similar to reduction method by avoiding generating CDF. It takes a group of data and reduces them to a single summated element. However, we obtain not only the total summation value, but the value saved in each thread which is useful for binary search in the next kernel. The final saved value of each thread is highlighted as in Fig. 5. With the summation saving in the last thread, the result of Eq. (10) is easily achieved.

At the beginning of resampling process, one uniform random number \( u \) is needed for the selection of one particle. With the regular-level summed value as Fig. 5 shows, binary search becomes operable.

Take 16 particles as an example, the first step of the selection starts from \( thread_7 \), which stores \( C_7^7 \) in the mid-thread of this array. Depending on the result, if the random number \( u \) is smaller than \( C_7^7 \), the value of \( u \) is not changed and the next comparison will be in \( thread_7 \) which stores \( C_7^8 \), the middle of the left half threads. On the other hand, if \( u \) is larger than \( C_7^7 \), \( u \) is subtracting the value in \( thread_7 \) and next comparison will be in \( thread_11 \) at the middle of the right half threads, which stores \( C_11^7 - C_11^7 \). For the next step, the value doesn’t change and goes to the left half when it is less than the compared value. For another possibility, \( u' \) is larger than the compared value, it would minus the value in that thread and go to the right half of this part. All the possibilities are shown in Fig. 5. For example, the selected destination ought to be \( thread_9 \). After first step comparison, next step would start at \( thread_11 \) with \( u' = u - value_7 \). \( thread_11 \) saves \( C_11^7 - C_7^7 \). Then after second time comparison, \( u'' = u' \) and goes to \( thread_5 \). While \( thread_11 \) saves \( C_11^7 - C_7^7 \), it then compares to \( thread_9 \) where it is smaller than the value in that thread. Then it is ensured that \( C_5^7 < u < C_7^7 \). From the example, it’s clear that with the generated random number comparing to values from a higher level to lower, selection of one particle can be achieved readily.

The time complexity of proposed binary searching oriented reweight is \( O(\log N) \). And the binary search process for each particle is equivalent and independent, which means that selection between particles could achieve parallelism. Thus, processing time won’t alter intensively while particle number changes. The concrete effects of this work are introduced in Sect. 4.

4. Experiment Result and Considerations

4.1 Experiment Environment

A heterogeneous platform is used to accelerate the algorithm. The experiment is based on GPU, the NVIDIA GEFORCE GTX 1080, and compares with the basic GPU work and the serial implementation based on the Intel i7-6700K, 4GHz, CPU. The program is written in C++ and CUDA C language.
4.2 Dataset and Evaluation Method

Videos used for the experiment is an official volleyball match (2014 Inter-High School Games of Men’s Volleyball held in Tokyo Metropolitan Gymnasium in Aug. 2014). For each sequence, synchronous videos taken from four cameras locating at corners of the court are used. Figure 6 shows the examples of evaluation frames and Fig. 7 shows the target ball image in the frames at the same time. Resolution of the videos is 1920 × 1080 and the frame rate is 60fps. Camera’s shutter speed is 1000 times per second to ensure less blur in videos. The evaluation dataset contains 118 sequences in total with no blur in the frames. And the average duration for each sequence is 7 seconds, varying from 3 seconds to 12 seconds.

Tracking accuracy evaluation method is same as Cheng’s [9]. A definition of the HIT is given, which presents a period between two consecutive ball hitting. When the ball is tracked continuously in one HIT, it is considered as success HIT. The success rate is calculated as below:

\[
\text{Success rate} = \frac{\sum \text{Success HIT}}{\sum \text{HIT}} \times 100\%.
\]  

(12)

For acceleration evaluation method, the time consumption \( T \) is considered as the main measurement. Speedup is calculated based on the equation below. The time consumption of data transmission between the CPU and the GPU, and the GPU kernel execution time is included. Initialization, reading image from files on CPU and outputting videos parts are not in our consideration.

\[
\text{Speedup} = \frac{T_{\text{CPU implementation}}}{T_{\text{GPU implementation}}}.
\]  

(13)

4.3 Experimental Result and Consideration

Table 1 presents the comparison of success rate and execution time between the CPU version of target algorithm [9], the basic GPU implemented version and the proposed one. Particle count is set as 1024 which is suitable for hardware implementation. As we can see, the success rate is similar, all accomplishing a success rate over 99%. The success rate in proposed system increases a little under the influence of the prediction of targets based on random numbers. The predicted position, based on Gaussian random numbers, is different in each run time. Thus, the result might be slightly different on same sequences even in the same algorithm. Also, compared with the generator in the CPU system, the generator of random numbers is slightly distinct by the GPU behavior in the proposed system, which causes a deviation to the result. With the acceptable deviation, the success rates in the CPU system, the basic GPU system and the proposed system are considered same basically. For the time consumption, our work shows a result as 3.05ms/f, compared with 75.1ms/f on CPU averagely, which has achieved a significant improvement.

Table 2 shows the time consuming for each kernel with 1024 particles.

| Time (ms/\( f \)) | Basic GPU Work | Proposal 2 | Proposal 2 + 3 | Proposal 2 + 3 + 1 |
|------------------|----------------|------------|----------------|------------------|
| K1: Prediction   | 0.4ms          | 0.4ms      | 0.4ms          | 0.4ms            |
| K2: Observation  | 4.3ms          | 1.5ms      | 1.5ms          | 1.5ms            |
| K3: Reweight     | 0.5ms          | 0.5ms      | 0.05ms         | 0.05ms           |
| K4: Resampling   | 0.3ms          | 0.3ms      | 0.1ms          | 0.1ms            |
| K5: CvtColor     | 1.3ms          | 1.3ms      | 1.3ms          | 1.3ms            |
| K6: RNG          | 0.3ms          | 0.3ms      | 0.3ms          | 0.3ms            |
| K1+K5            | 1.7ms          | 1.7ms      | 1.7ms          | 1.4ms            |
| K2+K3+K6        | 5.1ms          | 2.3ms      | 1.85ms         | 1.55ms           |
| Sum             | 7.1ms          | 4.3ms      | 3.65ms         | 3.05ms           |
1024 particles in the system. It also shows the effectiveness of each proposal by comparing with a basic GPU implemented work. The basic GPU work applied single stream flow, particle priority based threads allocation and sum reduction method[26] in reweight. In Table 2, proposal 1 is double stream system flow, proposal 2 stands for view priority threads allocation and proposal 3 represents binary search oriented reweight. It shows that K1 and K5 cost most of the time and the others contain less time complexity. Also, since synchronization occurs between two streams, kernels contain time overlapping has been listed and shown in the table as well. Due to the changing of observing image region range for likelihood calculation every frame, processing time of K2 floats between 1.3ms/f and 1.7ms/f. Thus, the corresponding minimum and maximum calculation time are 2.85ms/f and 3.25ms/f. The maximum time satisfies real-time processing for 60fps video sequences is 16.6ms/frame, thus the result of this work meet the requirement even with the worst performance.

When the algorithm is applied to different ball tracking, such as ping-pong [24] and tennis [25], larger particle count is needed to obtain high accuracy. Thus, we also analyzed the change of speedup with the particle count increasing, from 512 to 4096.

From Fig. 8, it’s easy to find out the difference between the CPU system, the basic GPU system and the proposed one. With particle count increasing, processing time on the CPU system floats violently from 40ms to over 200ms and increases from 7ms to over 15ms on the basic GPU system. On the other hand, time cost in the proposed CPU-GPU heterogeneous system hardly changes and keeps at 3ms. With the time of proposed system not changing, it is proved that parallelism during the tracking processing is achieved.

In Fig. 9 and Fig. 10, the proposed methods were evaluated within relative kernels. Figure 9 shows the comparison between the particle priority and proposed view priority based threads allocation. It’s easy to tell that the performance improvement is relatively high compared with the conventional one. Figure 10 presents that our work obtains a large performance promotion by reducing CDF generation and applying binary search in resampling compared with sum reduction method[26] and general binary search[27].

In Fig. 11, speedup time is compared between single stream and the proposed double stream system flow. When particle number increased, the speedup becomes considerable and the time cost doesn’t increase too much.

5. Conclusion

In this paper, a real-time 3D ball tracking system is proposed
based on the framework of chosen algorithm [9]. Firstly, double stream system flow obtains a tasks parallelism between kernels that has no data dependency. By operating kernels simultaneously, the system breaks the serial work flows limitation and shorten the executing time. Secondly, view priority based threads allocation is assigned in the observation task. For observation, coalesce memory access is obtained with the implementation of view priority based threads allocation and achieves memory accessing parallelism. Thirdly, binary search oriented reweight reduces the computation complexity. By avoiding generating the cumulative distribution function and applying binary search in an unordered array, computation complexity and serial processing are reduced. The proposed system maintains the high success rate among chosen tracking algorithm which is 99.23% while the processing time achieves 3.05ms/frame and a speed-up result at 24.62 times compared with CPU system program and 2.33 times compared with basic GPU implemented work. In the 60fps employed text sequences, the result fully satisfied the real-time target.

The future work will focus on the recovery method implementation. Though a high success rate result has been obtained, there are still chances that the tracking would fail. Due to the limitation of recording conditions, the ball sometimes moves out of the frame under some special situations. Also, in the game field, with the tracking targets’ differences and the noisy background changes, possibilities of tracking loss still exist. In that situation, the recovery algorithm should be in charge to find the target back and to retain the system result’s successfulness. A recovery method will help the system apply to more applications and different circumstances.

Acknowledgements

This work was supported by KAKENHI (16K13006) and Waseda University Grant for Special Research Projects (2017K-263.)

References

[1] R.A. Newcombe, A.J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, “KinectFusion: Real-time dense surface mapping and tracking,” In: 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pp.127–136, 2011.
[2] F.M. Cameras, J. Ren, J. Orwell, G. Jones, and M. Xu, Real-time 3D Soccer Ball Tracking[C], pp.829–838, 2004.
[3] N. Al-Najdawi, S. Tedmori, E. Edirisingle, and H. Bez, “An automated real-time people tracking system based on KLT features detection,” In: International Arab Journal of Information Technology, vol.9, no.9, pp.100–107, 2012.
[4] N. Peterfreund, “Robust tracking of position and velocity with Kalman snakes,” IEEE Trans. Pattern Anal. Mach. Intell., vol.21, no.6, pp.564–569, 1999.
[5] Y. Ohno, J. Miura, and Y. Shirai, “Tracking players and estimation of the 3D position of a ball in soccer games[C],” International Conference on Pattern Recognition, 2000, Proceedings, IEEE, pp.145–148, 2000.
[6] H.-T. Chen, W.-J. Tsai, S.-Y. Lee, and J.-Y. Yu, “Ball tracking and 3D trajectory approximation with applications to tactics analysis from single-camera volleyball sequences[J],” Multimedia Tools & Applications, vol.60, no.3, pp.641–667, 2012.
[7] G. Kitagawa, “Monte Carlo Filter and smoother for non-Gaussian nonlinear state space models,” Journal of Computational and Graphical Statistics, vol.5, no.1, pp.1–25, 1996.
[8] B. Ristic, S. Arulampalam, and N. Gordon, Beyond the Kalman Filter: Particle Filters for Tracking Applications, Artech House, 2004.
[9] X. Cheng, M. Honda, N. Ikoma, and T. Ikegami, “Anti-obclusion Observation Model and Automatic Recovery for Multi-view Ball Tracking in Sports Analysis,” In: 41st IEEE International Conference on Acoustics, Speech and Signal Processing, pp.1501–1505, 2016.
[10] R. Hartley and A. Zisserman, Multiple View Geometry in Computer Vision, In: Cam-bridge university press, 2003.
[11] S. Mittal and J.S. Vetter, “A Survey of CPU-GPU Heterogeneous Computing Techniques,” ACM Computing Surveys, vol.47, no.4, pp.1–35, 2015.
[12] Abi-Chahla, Fedy, Nvidia’s CUDA, The End of the CPU?, Tom’s Hardware, http://www.tomshardware.com/reviews/nvidia-cuda-gpu, 1954.html, Retrieved May 17, 2015.
[13] L. Claus, G. Panin, and A. Knoll, “A GPU-accelerated particle filter with pixel-level likelihood,” Vision, Modeling, and Visualization Conference 2008, Vmv 2008, Konstanz, Germany, October DBLP, pp.235–241, 2008.
[14] P. Li, “An efficient particle filter based tracking method using graphics processing unit (GPU),” J Signal Process Syst, vol.68, no.3, pp.317–332, 2012.
[15] Y. Hou, X. Cheng, and T. Ikegami, “Real-time 3D Ball Tracking with CPU-GPU Acceleration Using Particle Filter with Multi-command queues and Stepped Parallelism Iteration,” In: 2nd International Conference on Multimedia and Image Processing (ICMIP), pp.235–239, 2017.
[16] Z. Deng, Y. Hou, X. Cheng, and T. Ikegami, “Vectorized Data Combination and Binary Search Oriented Reweight for CPU-GPU Based Real-time 3D Ball Tracking,” Pacific-Rim Conference on Multime-
dia (PCM2017), Sept, 2017.
[17] J.A. Brown and D.W. Caspon, “A Framework for 3D Model-Based Visual Tracking Using a GPU-Accelerated Particle Filter[J],” IEEE Trans. Vis. Comput. Graphics, vol.18, no.1, pp.68–80, 2012.
[18] L. Murray, GPU acceleration of the particle filter: the Metropolis resampler[J], 2012.
[19] M.-A. Chao, C.-Y. Chu, C.-H. Chao, and A.-Y. Wu, “Efficient parallelized particle filter design on CUDA[C],” Signal Processing Systems, IEEE, pp.299–304, 2010.
[20] J. Thiagalingam, L. Kekempanos, and S. Maskell, “MapReduce Particle Filtering with Exact Resampling and Deterministic Run-time[J],” EURASIP J. Adv. Signal Process., 2017.
[21] M. Chitician, A. Simonetto, A.S. van Amesfoort, and T. Keviczky, “Distributed Computation Particle Filters on GPU Architectures for Real-Time Control Applications[J],” IEEE Trans. Control Syst. Technol., vol.21, no.6, pp.2224–2238, 2013.
[22] Paulius Mickevicius, Optimizing CUDA, https://mc.stanford.edu/cgi-bin/images/0/0a/M002-4.pdf
[23] P. Li, “An Efficient Particle Filter based Tracking Method Using Graphics Processing Unit (GPU)[J],” Journal of Signal Processing Systems, vol.68, no.3, pp.317–332, 2012.
[24] Z. Deng, X. Cheng, and T. Ikegami, “Ball-like Observation Model and Multi-peak Distribution Estimation Based Particle Filter for 3D Ping-pong Ball Tracking,” IAPR International Conference on Machine Vision Applications (MVA2017), pp.390–393, May 2017.
[25] Y. Wang, X. Cheng, N. Ikoma, M. Honda, and T. Ikegami, “Motion Prejudgment Dependent Mixture System Noise in System Model for Tennis Ball 3D Position Tracking by Particle Filter[C],” Joint, Inter-
national Conference on Soft Computing and Intelligent Systems, pp.124–129, 2016.
[26] S. Sengupta, M. Harris, Y. Zhang, and J.D. Owens, “Scan primitives
for GPU computing.” Proc. 22nd ACM Siggraph/eurographics Symposium on Graphics Hardware, Eurographics Association, pp.97–106, 2007.

[27] “bscich C binary search a sorted table,” The Open Group Base Specifications (7th ed.), The Open Group, 2013, Retrieved 28 March 2016.

[28] Z. Deng, Y. Hou, X. Cheng, and T. Ikenaga, “Multi-Peak Estimation for Real-Time 3D Ping-Pong Ball Tracking with Double-Queue Based GPU Acceleration,” IEICE Trans. Info. and Sys., vol.E101-D, no.5, pp.1251–1259, May 2018.

Yilin Hou received her B.E. degree in School of Automation from Beijing Institute of Technology, China, in 2016. She received her M.E. from the Graduate School of Information, Production, and Systems of Waseda University, Japan, in 2017. Her current research interests are sports analysis, computer vision and heterogeneous computing.

Ziwei Deng received her B.E. degree in School of Automation from Southeast University, China, in 2016. She received her M.E. from the Graduate School of Information, Production, and Systems of Waseda University, Japan, in 2017. Her current research interests are computer vision and heterogeneous computing.

Xina Cheng received her B.E. degree in School of Optoelectronics from Beijing Institute of Technology, China, in 2014. She received her M.E. and Ph.D. degrees from the Graduate School of Information, Production, and Systems of Waseda University, Japan, in 2015 and 2018. She is currently an assistant professor in Waseda University. Her current research interests are sports analysis and computer vision.

Takeshi Ikenaga received his B.E. and M.E. degrees in electrical engineering and Ph.D. degree in information & computer science from Waseda University, Tokyo, Japan, in 1988, 1990, and 2002, respectively. He joined LSI Laboratories, Nippon Telegraph and Telephone Corporation (NTT) in 1990, where he had been undertaking research on the design and test methodologies for high performance ASICs, a real-time MPEG2 encoder chip set, and a highly parallel LSI & system design for image-understanding processing. He is presently a professor in the system integration field of the Graduate School of Information, Production and Systems, Waseda University. His current interests are image and video processing systems, which covers video compression (e.g. H.265/HEVC, SHVC, SCC), video filter (e.g. super resolution, noise reduction, high-dynamic-range imaging), and video recognition (e.g. sport analysis, feature point detection, object tracking). He is a senior member of the Institute of Electrical and Electronics Engineers (IEEE), a member of the Institute of Electronics, Information and Communication Engineers of Japan (IEICE) and the Information Processing Society of Japan (IPSJ).