GPU-Accelerated Human Motion Tracking Using Particle Filter Combined with PSO

Boguslaw Rymut[2], Bogdan Kwolek[1], and Tomasz Krzeszowski[2]

[1] AGH University of Science and Technology, 30 Mickiewicza Av., 30-059 Kraków, Poland
bkw@agh.edu.pl

[2] Rzeszów University of Technology, W. Pola 2, 35-959 Rzeszów, Poland
brymut@prz.edu.pl

Abstract. This paper discusses how to combine particle filter (PF) with particle swarm optimization (PSO) to achieve better object tracking. Owing to multi-swarm based mode seeking the algorithm is capable of maintaining multimodal probability distributions and the tracking accuracy is far better than accuracy of PF or PSO. We propose parallel resampling scheme for particle filtering running on GPU. We show the efficiency of the parallel PF-PSO algorithm on 3D model based human motion tracking. The 3D model is rasterized in parallel and single thread processes one column of the image. Such level of parallelism allows us to efficiently utilize the GPU resources and to perform tracking of the full human body at rates of 15 frames per second. The GPU achieves an average speedup of 7.5 over the CPU. For marker-less motion capture system consisting of four calibrated cameras, the computations were conducted on four CPU cores and four GTX GPUs on two cards.

1 Introduction

Particle filtering is a widely used framework for visual object tracking since it offers the flexibility to handle non-linearity and non-normality of the object models. However, the huge number of particles, which is needed in applications like 3D model based articulated motion tracking limits their wide application, particularly if the tracking should be done in real-time. In typical applications of particle filters, the resampling is needed because of undesirable degeneracy problem, where all but one particle will have negligible weight after a few iterations. The widely used resampling algorithms in the particle filters are sequential in essence [5]. In [6] a shared-memory resampling algorithm was proposed. In the discussed work a left and right boundary was employed in the systematic resampling [1]. The two introduced variables remove the data dependency that keeps the systematic resampling serial. In our approach to parallel resampling we utilize the binary search algorithm [8] for selecting the particle indices.

To reduce the number of the particles, Deutscher and Reid [4] developed an annealed particle filter (APF), which adopts an annealing scheme to achieve

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-319-02895-8_64 better concentration of the particles and shift them towards the modes of the probability distribution. Additionally, a crossover operation was proposed to achieve better diversity of the particles. Compared with the ordinary particle filter (PF), the annealed particle filter greatly enhances the tracking accuracy. However, since the particles do not exchange information they have reduced capability of focusing the search on the promising areas. In contrast, the particle swarm optimization (PSO) [9] has better capability of exploration of the search space owing to combining the local search (by self experience) and global one (by neighboring experience).

In this paper we discuss how to combine a particle filter with a particle swarm optimization to reduce the number of particles that is needed to achieve the desirable tracking accuracy. Owing to multi-swarm based mode seeking the tracker is capable of maintaining multimodal probability distributions and the tracking accuracy is far better than accuracy of PF or PSO. We show the efficiency of our parallel PF-PSO algorithm on 3D model based human motion tracking.

2 GPU Computing

The programming of GPU has been considerably simplified through introducing CUDA framework by NVIDIA. In CUDA the parallel portions of an application are executed on GPU as kernels. A CUDA kernel is executed by an array of threads. Blocks of threads are organized into a one-dimensional or two-dimensional or three-dimensional grid of thread blocks. Blocks are mapped to multiprocessors and each thread is mapped to a single core. A warp is a group of threads within a block that are launched together and usually execute together. When a warp is selected for execution, all active threads execute the same instruction but operate on different data. A unique set of indices is assigned to each thread to determine to which block it belongs and its location inside it.

In order to obtain the best computing performance on GPU, we have to keep all processors occupied and hide memory latency. In order to achieve this aim, CUDA supports running hundred or thousands of lightweight threads in parallel. The benefit of having multiple blocks per multiprocessor is that the scheduling hardware is capable to swap out a block that is waiting on a high-latency instruction and replace it with a block that has threads ready to execute. The context switch is very fast because everything is stored in registers and thus there is almost no data movement. To achieve good performance both high density of arithmetic instructions per memory access as well as several hundreds of threads per block are needed. This permits the GPU to execute arithmetic instructions while certain threads are waiting for access to the global memory.

The global memory resides off chip and has a large data bus resulting in very large bandwidth. It is accessible from different blocks. Its latency can be hidden by careful design of control flow and adequate design of kernels. The shared memory resides on chip. It is shared between all processors of a multi-processor block, but two threads from different blocks cannot cooperate via shared memory. Its latency is several times shorter than the latency of the global memory.

3 3D Model-Based Human Motion Tracking

The articulated model of the human body has a form of a kinematic chain consisting of 11 segments. The 3D model is constructed using truncated cones (frustums) that represent the pelvis, torso, head, upper and lower arm and legs. The model has 26 DOF and its configuration is determined by position and orientation of the pelvis in the global coordinate system and the relative angles between the limbs. Each truncated cone is parameterized by the center of base circle A, center of top circle B, bottom radius $r1$, and top radius $r2$. Given the 3D camera location C and 3D coordinates A and B, the plane passing through the points is determined. Since the vectors AB and AC lie in the plane, their cross product, which is perpendicular to the plane of AB and AC, is the normal. The normal is used to determine the orientation of the trapezoid to be projected into 2D plane. Each trapezoid of the model is projected into 2D image of each camera via modified Tsai's camera model. The projected image of the trapezoid is obtained by projecting the corners and then a rasterization of the triangles composing the trapezoid. The main task of a filling rasterizer is to seek all pixels that are covered by a triangle. Though projecting all truncated cones we obtain the image representing the 3D model in a given configuration.

For each set of frames acquired simultaneously by the synchronized cameras, the 3D human pose is reconstructed through matching the current image observations with the projected human body model into each camera view. In most of the approaches to articulated object tracking a background subtraction algorithms are employed to extract the subject undergoing tracking. Additionally, image cues such as edges, ridges and color are often employed to improve the extraction of the person. In the presented approach the human silhouette is extracted via background subtraction. Afterwards, the edges are located within the extracted silhouette. Finally, the edge distance map is extracted. The matching score reflects (i) matching ratio between the extracted silhouette and the projected 3D model and (ii) the normalized distance between the model's projected edges and the closest edges in the image [11]. The objective function of all cameras is the sum of such matching scores.

3.1 Rasterization of the 3D Model

In PF-based or PSO-based 3D motion tracking a single particle represents a hypothesis about possible subject pose. In such approaches the most computationally and time demanding operation is evaluation of the particles' score. Since every particle represents a hypothesis about possible subject pose the CPU-time needed for the rasterizing the 3D models for all particles can be considerable. As we already mentioned, the image of the projected model is obtained by rasterization of the triangles composing the trapezoids. In our approach the triangles that are closer to the camera are rasterized first. During the rendering of a next triangle the algorithm checks if a triangle has already been rasterized. If yes, it skips the rasterization, i.e. the triangle already rasterized is not painted over. Such an operation is performed for each triangle.

In the evaluation of the particles' score the projected and rasterized 3D model is matched with the current image observation. The fitness score depends on the degree of overlap between the extracted silhouette in the current image and the projected and rasterized 3D model in the hypothesized pose. The overlap degree is calculated through checking the overlap from the silhouette to the rasterized model as well as from the rasterized model to the silhouette. The larger the degree overlap is, the larger is the fitness value. The objective function reflects also the normalized distance between the model's projected edges and the closest edges in the image. It is calculated on the basis of the edge distance map [11].

3.2 Parallelization of Model Rasterization

In the evaluation of particles scores we employ two kernels. In the first one the 3D models are projected into 2D image of each camera. In the second one we rasterize the models and evaluate the objective functions. In our approach, in every block we rasterize the model in the pose represented by a single particle as well as we calculate its fitness score. Thus, the number of blocks is equal to the number of the particles, see Fig. 1. Each thread is responsible for rasterizing the model in single column and summing the fitness values of the pixels in that column. The number of threads in each block is equal to the image width, whereas the number of running threads in each block is equal to the number of cores per multiprocessor, see Fig. 1.

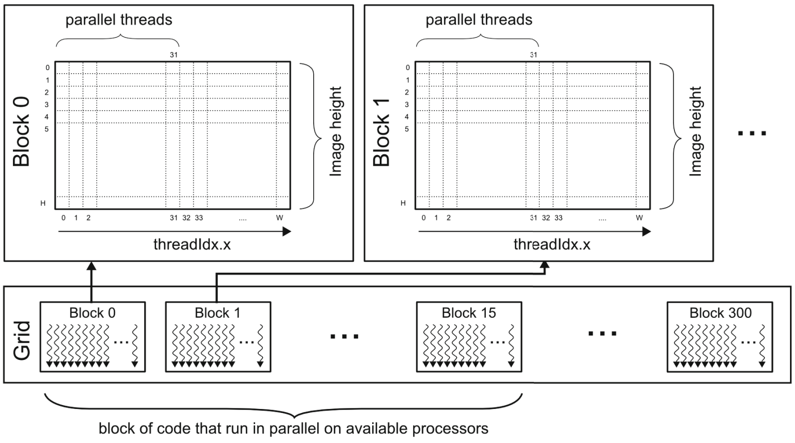

Fig. 1. Parallelization of the cost function

The cost values of the objective function are summed using parallel reduction. The results from each column of the threaded block are stored in the shared memory. In the next stage, $W/2$ consecutive threads determine the sums of the two adjacent memory cells of the shared memory and then store the results in the shared memory. The next iteration employs $W/4$ threads to add the results of the previous iteration, and so on. In order to speedup the triangle rasterization, for each triangle a rasterization area is determined. The rasterization is then only performed for pixels belonging to such a constrained area.

4 Parallel PSO for Object Tracking

Particle Swarm Optimization [9] is a bioinspired meta-heuristic for solving complex global optimization problems. The PSO is initialized with a group of random particles (hypothetical solutions) and then it searches for optima by updating all particles locations. The particles move through the solution space and undergo evaluation according to some fitness function. Each particle iteratively evaluates the candidate solutions and remembers the personal best location with the best objective value found so far, making this information available to its neighbors. Particles communicate good positions to each other and adjust their own velocities and positions taking into account such good locations.

In the ordinary PSO algorithm the update of particle's velocity and position can be expressed by the following equations:

$$v_j^{(i)} \leftarrow w v_j^{(i)} + c_1 r_{1,j}^{(i)} (p_j^{(i)} - x_j^{(i)}) + c_2 r_{2,j}^{(i)} (p_{\mathrm{g},j} - x_j^{(i)}) \tag{1}$$

$$x_j^{(i)} \leftarrow x_j^{(i)} + v_j^{(i)} \tag{2}$$

where w is the positive inertia weight, $v_j^{(i)}$ is the velocity of particle i in dimension j, $r_{1,j}^{(i)}$ and $r_{2,j}^{(i)}$ are uniquely generated random numbers with the uniform distribution in the interval [0.0, 1.0], c_1, c_2 are positive constants, $p^{(i)}$ is the best position that the particle i has found so far, p_g denotes best position that is found by any particle in the swarm.

The velocity update equation (1) has three main components. The first component, which is often referred to as inertia models the particle's tendency to continue the moving in the same direction. In effect it controls the exploration of the search space. The second component, called cognitive, attracts towards the best position $p^{(i)}$ previously found by the particle. The last component is referred to as social and attracts towards the best position p_g found by any particle. The fitness value that corresponds $p^{(i)}$ is called local best $p_{\mathrm{best}}^{(i)}$, whereas the fitness value corresponding to p_g is referred to as g_{best}.

Our parallel PSO algorithm for object tracking consists of five main phases, namely initialization, evaluation, *p_best*, *g_best*, update and motion, see Fig. 2. Before each optimization cycle, in the initialization stage an initial position $x^{(i)}$ and velocity $v^{(i)}$ is assigned to each particle. In the evaluation phase the fitness value of each particle is calculated using a cost function. In the *p_best* stage the determining of $p_{\mathrm{best}}^{(i)}$ as well as $p^{(i)}$ takes place. The operations mentioned above are computed in parallel using available GPU resources. Afterwards, the g_{best} and its corresponding p_g are calculated in a sequential task. Finally, the update stage is done in parallel. That means that in our implementation we employ

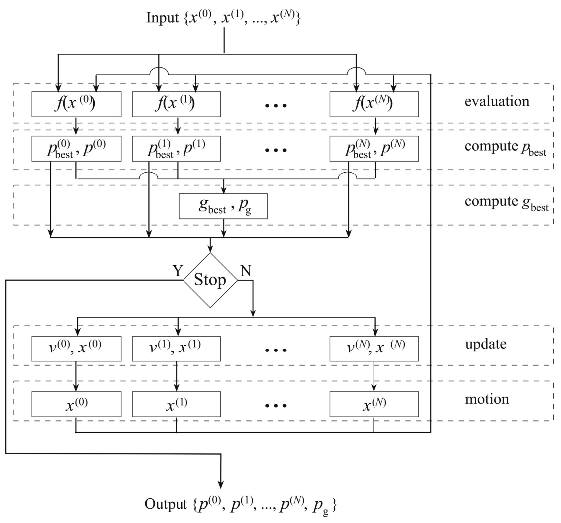

Fig. 2. Decomposition of synchronous particle swarm optimization algorithm on GPU

the parallel synchronous particle swarm optimization. The synchronous PSO algorithm updates all particle velocities and positions at the end of each iteration. In contrast to synchronous PSO the asynchronous algorithm updates particle positions and velocities continuously using currently accessible information.

In order to decompose an algorithm onto GPU we should identify data-parallel portions of the program and isolate them as CUDA kernels. In the initialization kernel we generate pseudo-random numbers using the curand library provided by the CUDA^{TM} SDK. On the basis of the uniform random numbers we generate normally distributed pseudorandom numbers using Box Mueller transform based on trigonometric functions [3]. The normally distributed random numbers are generated at the beginning of each frame to initialize the particles' velocities. Then the uniform random numbers r_1, r_2 for the optimal pose seeking are generated. This means that for every particle we generate $2 \times D \times K$ normally distributed random numbers, where D is dimension and K denotes the maximum number of iterations. They are stored in the memory and then used in the update kernel. At this stage the computations are done in $\lceil N/(2 \times W) \rceil$ blocks and W threads on each of them, where W denotes the number of cores per multiprocessor. In the compute p_{best} kernel and the update kernel the number of blocks is equal to $\lceil N/W \rceil$, whereas the number of threads in each block is equal to W. In the update kernel we constrain the velocities of the particles to the assumed maximal velocity values. In the motion stage the model's bone hierarchy is recursively traversed and the internal transformation matrices are updated according to the state vector of the particle.

4.1 Fitness Score for Particle Swarm Optimization

The fitness score for i-th camera's view is calculated on the basis of following expression: $f^{(i)}(\mathbf{x}) = 1 - ((f_1^{(i)}(\mathbf{x}))^{w_1} \cdot (f_2^{(i)}(\mathbf{x}))^{w_2})$, where w denotes weighting coefficients that were determined experimentally. The function $f_1^{(i)}(\mathbf{x})$ reflects the degree of overlap between the extracted body and the projected 3D model into 2D image corresponding to camera i. The function $f_2^{(i)}(\mathbf{x})$ reflects the edge distance-based fitness in the image from the camera i. The objective function for all cameras is determined according to the following expression: $f(\mathbf{x}) = \frac{1}{4}\sum_{i=1}^{4} f^{(i)}(\mathbf{x})$. The images acquired from the cameras are processed on CPU and then transferred onto the device.

5 Parallel PF for Object Tracking

The particle filter simulates the behavior of the dynamical system. Each sample predicts future behavior of the system in a Monte-Carlo fashion, and the samples that match the observed system behavior are kept, whereas ones that are unsuccessful in predicting tend to die out. The evolution of the state of the target as well as its measurement process is modeled by a set of (possibly non-linear) equations perturbed by (possibly non-Gaussian) i.i.d. noise:

$$\mathbf{x}_k = f_k(\mathbf{x}_{k-1}, \mathbf{v}_k) \tag{3}$$

$$\mathbf{z}_k = h_k(\mathbf{x}_k, \mathbf{n}_k) \tag{4}$$

where $\mathbf{x}_k$ denotes the state of the target at discrete time k, $\mathbf{v}_k$ is the process noise vector, $\mathbf{z}_k$ is the measurement vector, and $\mathbf{n}_k$ is the measurement noise vector. The aim is to estimate the distribution of the target state given all the previous measurements, that is, $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$, where $\mathbf{z}_{1:k-1} = \{\mathbf{z}_1, \ldots, \mathbf{z}_{k-1}\}$. The recursive Bayesian filter first calculates the *a priori* density $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ using the system model and then evaluates *a posteriori* density $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ given the new measurement.

In the PF, the distribution $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ is approximated by a set of M particles $\{\mathbf{x}_{k-1}^i\}_{i=1\ldots M}$ and associated weights $\{w_{k-1}^i\}_{i=1\ldots M}$ in the following manner: $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) \approx \sum w_{k-1}^i \delta(\mathbf{x}_k - \mathbf{x}_{k-1}^i)$, where $w_k^i \propto w_{k-1}^i \frac{p(\mathbf{z}_k|\mathbf{x}_k^i)p(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i)}{q(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i,\mathbf{z}_k)}$, whereas $\sum w_{k-1}^i = 1$ and $\delta(\cdot)$ denotes the Kronecker delta function. The term $q(\mathbf{x}_k^i|\mathbf{x}_{k-1}^i, \mathbf{z}_k)$ stands for an importance density, which is typically obtained by approximating $p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{z}_k)$ with a Gaussian distribution, or alternatively by using $p(\mathbf{x}_k|\mathbf{x}_{k-1})$.

One of the practical difficulties that is associated with particle filters is degeneration of the particle population after a few iterations because weights of several particles are negligible, and, eventually, only a very small number of particles contributes to the posterior distribution. To mitigate this problem the resampling should be used in order to eliminate particles with low importance weights and multiply particles with high importance weights. Resampling can be carried out at every iteration or only when a substantial amount of degeneracy is observed [5]. The algorithm can be expressed by the pseudo-code:

1. For $i = 1, 2, \ldots, M$ sample or propose particles using $p(\mathbf{x}_k|\mathbf{x}_{k-1})$
2. For $i = 1, 2, \ldots, M$ calculate the weights, $\tilde{w}_k^i = w_{k-1}^i p(\mathbf{z}_k|\mathbf{x}_k^i)$
3. Normalize the weights w_k^i using $\tilde{w}_k^i$
4. Calculate the state estimates, $\hat{\mathbf{x}}_k = \sum_{i=1}^{M} w_k^i \mathbf{x}_k^i$
5. Resample $\{\mathbf{x}_k^i, w_k^i\}$ to get new set of particles $\{\mathbf{x}_k^j, w_k^j = 1/M\}$

The resampling step can be expressed as follows:

1. Given the normalized weight, calculate an array of the cumulative sum of the weights.
2. Randomly generate a number and determine which range in that cumulative weight array to which the number belongs.
3. The index of that range would correspond to the particle that should be selected.
4. Repeat until the desired number of samples is selected.

The cumulative sum of the weights has been computed in parallel using all-prefix-sum operation [2]. The all-prefix-sums operation takes a binary associative operator $\oplus$, and an ordered set of n elements $a_0, a_1, \ldots, a_{n-1}$ and returns the ordered set $a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus, \ldots, \oplus\, a_{n-1})$. The cumulative sum was calculated using CUDA implementation of all-prefix-sums operation [7].

Given the cumulative sum of the weights, we randomly generate a number and then determine which range in that cumulative weight array to which the

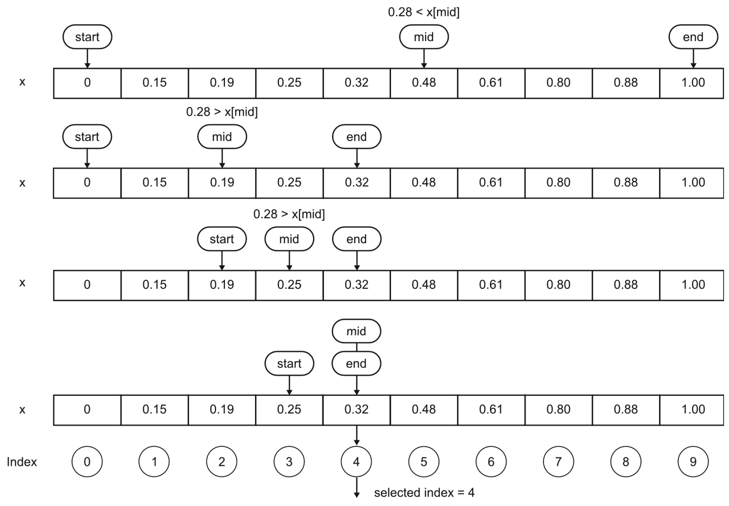

Fig. 3. Binary Search Algorithm - selecting indices based on uniform random numbers. Search for index corresponding to value 0.28 (generated by RNG).

number belongs. The search for the particle index corresponding to the generated random number was done in parallel using the binary search operation from Thrust parallel algorithms library [8], see also illustrative example on Fig. 3.

5.1 Observation Model of Particle Filter

In a particle filter the observation model describes the likelihood of a given observation given the considered object state. It assumes the following form: $p(\mathbf{z}_k|\mathbf{x}_k) = \exp(-f(x)^2/(2\sigma^2))$.

6 Parallel PF-PSO for Object Tracking

The strength of particle filter lies in their ability to represent multi-modal probability distributions and to maintain multiple hypotheses about target state. However, the number of particles that is required to adequately approximate the underlying probability distribution in the pose space might be huge. To mitigate this limitation, Deutscher and Reid [4] developed an annealed particle filter (APF).

Compared with the ordinary particle filter, the annealed particle filter greatly enhances the tracking accuracy. However, considerable number of particles is still required. Because the particles do not exchange information they have reduced capability of focusing the search on the promising areas. In contrast, owing to combining the local search (by self experience) and global one (by neighboring experience), the PSO algorithm has better capability of exploration of the search space. In [10] we demonstrated that a particle filter combined with a particle swarm optimization is better than each of them in terms of quality of tracking. In this work we present a modified PF-PSO, which has better capability of dealing with multimodal distributions. On the other hand, through the use of multiple swarms the diversity of PSO is also better.

In order to utilize the advantages of both algorithms, to emphasize their complementarities, and in particular to achieve real-time 3D motion tracking of full

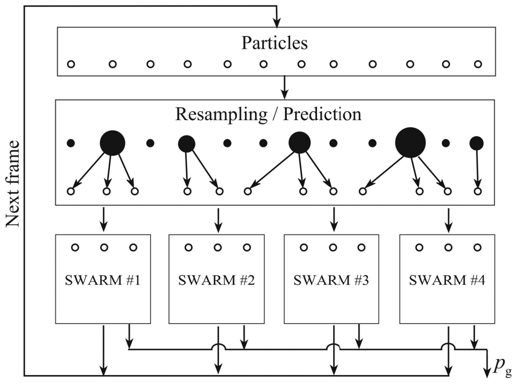

Fig. 4. PF-PSO algorithm

human body we elaborated a parallel PF-PSO algorithm. A flowchart of the algorithm is depicted in Fig. 4. At the beginning of each frame a PF with parallel resampling is executed.

After the resampling a diversification of the particles takes place. The particles are then assigned to four swarms and each self-optimizing swarm executes a specified number of iterations. In consequence, owing to multi-swarm optimization the algorithm is able to maintain the multimodal distributions. In the last stage the particles are propagated over time to cover the promising object poses in the next frame.

7 Experimental Results

The proposed PF-PSO algorithm has been evaluated on image sequences from [11]. They were acquired by four calibrated and synchronized cameras with 25 frames per second. The first pair of the cameras was approximately perpendicular to the second camera pair. The input images of size 1920×1080 were resized to 480×270 resolution. A commercial motion capture (MoCap) system from Vicon Nexus was employed to provide the ground truth data. The MoCap system delivers the data with rate of 100 Hz. The synchronization between MoCap and multi-camera system was accomplished using hardware from Vicon Giganet Lab. In the above mentioned work we demonstrated sample images with walking actors as well as the layout of the scene.

The execution time of the proposed algorithm was measured on a PC computer equipped with Intel Xeon X5690 3.46 GHz CPU (6 cores), with 8 GB RAM, and two NVidia GTX 590 graphics cards, each with 16 multiprocessors and 32 cores per multiprocessor. Each card has two GTX GPUs, each equipped with 1536 MB RAM and 48 KB shared memory per multiprocessor. Table 1

Table 1. Computation time [ms] and speedup for marker-less MoCap system consisting of 4 cameras. The times are for images from single cameras' shot.

			Seq. 1			Seq. 2		
	#particles	it.	GTX590	CPU	speedup	GTX590	CPU	speedup
PF	1000		14.5	143.1	9.9	15.2	141.6	9.3
	2000		26.7	264.6	9.9	26.3	263.0	10.0
	3000		40.0	387.1	9.8	39.6	388.5	9.8
	4000		52.2	515.8	9.9	54.5	517.7	9.5
PSO	100	10	30.7	138.7	4.5	30.5	139.0	4.6
	200	10	41.5	253.3	6.1	42.3	254.2	6.0
	300	10	58.5	368.0	6.3	58.8	370.0	6.3
	400	10	70.8	484.0	6.8	72.1	482.7	6.7
PF-PSO	100	10	33.3	139.3	4.2	31.2	138.5	4.4
	200	10	45.8	255.5	5.6	45.4	253.5	5.6
	300	10	57.7	371.3	6.4	55.5	367.7	6.6
	400	10	67.8	487.1	7.2	68.5	484.1	7.1

shows the computation time that we obtained on CPU and GPU for a MoCap configuration with four cameras. The processing times are in milliseconds. It contains the time needed for processing four images from single cameras' shot.

We compared the execution time of a generic particle filter (PF), synchronous Particle Swarm Optimization (PSO) and particle filter combined with PSO (PF-PSO) as well as the speedup of the GPU over GPU. The number of evaluations of the cost function by the considered algorithms was the same. For instance, the PF consisting of 4000 particles was compared with PSO consisting of 400 particles and executing 10 iterations. In the evaluation tests we used 4 CPU cores and 4 GPUs, i.e. on each GPU or CPU we processed the images from a single camera. The input images were preprocessed off-line and then transferred frame by frame to the GPU. As we can notice, the speedup of the algorithm executed on GPU over the CPU implementation is considerable. The speedup of PF does not vary much with the number of particles. For PSO and PF-PSO the speedup grows with the number of the particles. The speedup achieved by the PF is slightly better in comparison to speedup achieved by PSO and PF-PSO. As we can see, the full body motion can be tracked at frames larger than 15 frames by PSO or PF-PSO consisting of 300 particles and executing 10 iterations.

In Tab. 2 are shown the tracking accuracies that were obtained by the discussed algorithms. The errors were calculated using 39 markers. For each frame they were computed as average Euclidean distance between individual markers and the recovered 3D joint locations. For each sequence they were then averaged over ten runs of the algorithm with unlike initializations. As we can observe, the PF-PSO allows us to achieve superior tracking accuracy. Only for a PSO consisting of 100 particles and executing 10 iterations the tracking accuracy of the PSO is better than the accuracy of the PF-PSO. Moreover, the standard deviation achieved by the PF-PSO is far smaller in comparison to the standard deviation

Table 2. Average errors for $M = 39$ markers in two image sequences acquired by four synchronized and calibrated cameras

			Seq. 1		Seq. 2	
	#particles	it.	error [mm]	std. dev. [mm]	error [mm]	std. dev. [mm]
PF	1000		98.9	60.5	105.3	69.9
	2000		74.1	46.1	92.9	61.6
	3000		72.3	41.5	75.3	49.0
	4000		65.7	35.7	66.8	40.5
PSO	100	10	51.1	37.2	63.0	40.3
	200	10	59.1	49.6	65.0	48.2
	300	10	59.8	50.0	62.7	45.4
	400	10	55.2	44.1	53.9	37.5
PF-PSO	100	10	58.8	52.5	71.5	54.9
	200	10	49.4	34.8	58.2	39.0
	300	10	48.6	29.3	58.7	40.8
	400	10	49.9	36.0	53.9	35.5

achieved by the PF and the PSO. This means, that the PF-PSO algorithm copes better with the multimodal distributions than PSO algorithm.

8 Conclusions

We presented a PF-PSO algorithm, which has superior tracking accuracy to PF and PSO. Owing to multi-swarm based optimization the algorithm is capable of maintaining multimodal probability distributions. We proposed a parallel resampling scheme for particle filtering. The tracking of full human body can be performed at frame-rates of 16 frames per second using a two high-end graphics cards and images acquired by four cameras. The algorithm running on GPU is about 7.5 times faster in comparison to algorithm running on CPU. The average error is below 55 mm.

Acknowledgment. This work has been partially supported by the National Science Center (NCN) within the research project N N516 483240 and Ministry of Science and Higher Education within the grant U-206/DS/M.

References

1. Arulampalam, M., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. Trans. Sig. Proc. 50(2), 174–188 (2002)
2. Blelloch, G.E.: Prefix sums and their applications. Tech. Rep. CMU-CS-90-190, School of Computer Science, Carnegie Mellon University (November 1990)
3. Box, G.E.P., Muller, M.E.: A note on the generation of random normal deviates. The Annals of Mathematical Statistics 29(2), 610–611 (1958)
4. Deutscher, J., Blake, A., Reid, I.: Articulated body motion capture by annealed particle filtering. In: IEEE Int. Conf. on Pattern Recognition, pp. 126–133 (2000)
5. Doucet, A., Godsill, S., Andrieu, C.: On sequential Monte Carlo sampling methods for bayesian filtering. Statistics and Computing 10(1), 197–208 (2000)
6. Gong, P., Basciftci, Y.O., Ozguner, F.: A parallel resampling algorithm for particle filtering on shared-memory architectures. In: IEEE Int. Parallel and Distributed Processing Symposium, pp. 1477–1483. IEEE Computer Society (2012)
7. Harris, M., Sengupta, S., Owens, J.D.: Parallel prefix sum (scan) with CUDA. In: Nguyen, H. (ed.) GPU Gems 3. Addison Wesley (August 2007)
8. Hoberock, J., Bell, N.: Thrust: A parallel template library, version 1.3.0 (2010), `http://www.meganewtons.com/`
9. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. of IEEE Int. Conf. on Neural Networks, pp. 1942–1948. IEEE Press, Piscataway (1995)
10. Krzeszowski, T., Kwolek, B., Wojciechowski, K.: Articulated body motion tracking by combined particle swarm optimization and particle filtering. In: Bolc, L., Tadeusiewicz, R., Chmielewski, L.J., Wojciechowski, K. (eds.) ICCVG 2010, Part I. LNCS, vol. 6374, pp. 147–154. Springer, Heidelberg (2010)
11. Kwolek, B., Krzeszowski, T., Gagalowicz, A., Wojciechowski, K., Josinski, H.: Real-time multi-view human motion tracking using particle swarm optimization with resampling. In: Perales, F.J., Fisher, R.B., Moeslund, T.B. (eds.) AMDO 2012. LNCS, vol. 7378, pp. 92–101. Springer, Heidelberg (2012)