Real-Time Multiview Human Body Tracking using GPU-accelerated PSO

Boguslaw Rymut[2] and Bogdan Kwolek[1]

[1] AGH University of Science and Technology, 30 Mickiewicza Av., 30-059 Krakow, Poland
bkw@agh.edu.pl
[2] Rzeszów University of Technology, W. Pola 2, 35-959 Rzeszów, Poland
brymut@prz.edu.pl

Abstract. This paper presents our approach to 3D model-based human motion tracking using a GPU-accelerated particle swarm optimization. The tracking involves configuring the 3D human model in the pose described by each particle and then rasterizing it in each particle's 2D plane. In our implementation, we launch one independent thread for each column of each 2D plane. Such a parallel algorithm exhibits the level of parallelism that allows us to effectively utilize the GPU resources. Owing to such task decomposition the tracking of the full human body can be performed at rates of 15 frames per second. The GPU achieves an average speedup of 7.5 over the CPU. The speedup that achieves the GPU over CPU grows with the number of the particles. For marker-less motion capture system consisting of four calibrated and synchronized cameras, the efficiency comparisons were conducted on four CPU cores and four GTX GPUs on two cards.

Keywords: GPGPU, real-time computer vision, human motion capture

1 Introduction

In the early years of computer graphics, the GPU could only be programmed through a graphics rendering interface. Over the years, the GPU has evolved from a highly specialized graphics processor to a versatile and highly programmable architecture that can perform a wide range of data-parallel operations. The GPU architectures benefit from massive fine-grained parallelization, as they are able to execute as many as thousands of threads concurrently. Recently, many research papers reported that general purpose GPUs (GPGPUs) are capable to obtain significant speedups compared to current homogeneous multicore systems in the same price range. These scientific reports initiated a passionate debate on the limits of GPU-supported acceleration for various classes of applications [1]. A comparison of 14 various implementations showed speedups from 0.5× to 15× (GPU over CPU). The experiment was made with Intel Core i7 and NVidia GTX 280. There is a common agreement that in order to achieve satisfactory performance the algorithms to be executed on GPU should be carefully designed.

CPUs are still the most frequently used hardware for image processing. On the other hand, image processing algorithms are good candidates for GPU implementation, since the parallelization is naturally provided by per-pixel operations. Many research studies confirmed this by showing GPU acceleration of many image processing algorithms [2]. A recent study [3] reports a speedup of 30 times for low-level algorithms and up to 10 times for high-level functions.

Non intrusive human body tracking is a key issue in user-friendly human-computer communication. This is one of the most challenging problems in computer vision being at the same time one of the most computationally demanding tasks. Particle filters are typically employed to achieve articulated motion tracking. Several improvements of ordinary particle filter were done to achieve fast and reliable articulated motion tracking [4] as well as to obtain the initialization of the tracking [5]. 3D motion tracking can be perceived as dynamic optimization problem. Recently, particle swarm optimization (PSO) [6] has been successfully applied to achieve human motion tracking [7, 8]. The motion tracking is achieved by a sequence of static PSO-based optimizations, followed by re-diversification of the particles to cover the possible poses in the next time step.

There are only a few publications that discuss the implementation details of the PSO on GPU. In [9], an approach that restricts the communication of a particle to its two closest neighbors and thus limits the communication between threads was proposed. The authors of [10] compared three different variants of the PSO on GPU, but only parallelized the cost function. In [11], a multi-swarm PSO algorithm was used to achieve a high degree of parallelism. In [7] an approach to PSO-based full body human motion tracking on GPU and using single camera has been proposed. The 3D model with 26 DOF was constructed using cuboids, which were projected into 2D plane and then rendered in parallel. A single thread was responsible for comparing images containing the projected model and the extracted person. The tracking of the full human body was performed with 5 frames per second, whereas the speedup of GTX280 over a CPU was about 15. A common approach to parallelize the PSO consists in executing a local swarm on every processor while optimizing the communication between the swarms. Mussi et al. [8] proposed an approach to articulated human body tracking from multi-view video using PSO running on GPU. Their implementation is far from real-time and roughly requires 7 seconds per frame. Recently, in [12] a framework for 3D model-based visual tracking using a GPU-accelerated particle filter has been presented. A hand was tracked using both synthetic and real videos. The authors reported a speedup of 9.5 and 14.1 against a CPU for image resolution of 96×72 and 128×96 using 900 and 1296 particles, respectively.

In this work we present an approach that effectively utilizes the advantages of modern graphics card hardware to achieve real-time full body tracking using a 3D human model. The motion tracking was accomplished by a PSO algorithm running on a GPU. The presented approach to 3D articulated human tracking follows the Black Box Optimization paradigm [13], according to which the search processes/particles investigate the hypothesis space of a model state in order to identify the hypothesis that optimally fit a set of observations.

2 GPU Computing

CUDA is a parallel computing platform and programming model invented by NVIDIA. Each function that is executed on the device is called a kernel. A CUDA kernel is executed by an array of threads. Blocks of threads are organized into one, two 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.

GPUs offer best performance gains when all processing cores are utilized and memory latency is hidden. In order to achieve this aim, it is common to launch a CUDA kernel with hundreds or thousands of threads to keep the GPU busy. 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 the GPU does not have to store the state, as the CPU does when switching threads between being active and inactive. Thus, it is advantageous to have both high density of arithmetic instructions per memory access as well many more resident threads than GPU cores so that memory latency can be hidden. This permits the GPU to execute arithmetic instructions while certain threads are waiting for access to the global memory.

Memory latency can be hidden by careful design of control flow as well as adequate design of kernels. The kernels can employ not only the global memory that resides off chip, but also they can use shared memory that resides on chip. This memory is shared between all the cores of stream multiprocessor. Its latency is several times shorter than the latency of the global memory. Threads that are executing within the same block can cooperate using it, but threads from different block cannot cooperate via shared memory.

3 Parallel PSO for Object Tracking

Particle Swarm Optimization (PSO) [6] is a bio-inspired meta-heuristic for solving complex optimization problems. The PSO is initialized with a group of random particles (hypothetical solutions) and then 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. Additionally each particle utilizes a best value, which can be:

- a global best that is immediately updated when a new best position is found by any particle in the swarm

- neighborhood best where only a specific number of particles is affected if a new best position is found by any particle in the sub-population

Typically, a swarm topology with the global best converges faster since all particles are attracted simultaneously to the best part of the search space. Neighborhood best permits parallel exploration of the search space and decreases the susceptibility of falling into local minima. However, such a topology slows down the convergence speed. Taking into account the faster convergence the topology with the global best has been selected for parallel implementation.

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, 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}. The ordinary PSO algorithm can be expressed by the following pseudo-code:

1. Assign each particle a random position in the problem hyperspace.
2. Evaluate the fitness function for each particle.
3. For each particle i compare the particle's fitness value with its $p_{\mathrm{best}}^{(i)}$.
   If the current value is better than the value $p_{\mathrm{best}}^{(i)}$, then set this value as the $p_{\mathrm{best}}^{(i)}$ and the current particle's position $x^{(i)}$ as $p^{(i)}$.
4. Find the particle that has the best fitness value g_{best}.
5. Update the velocities and positions of all particles according to (1) and (2).
6. Repeat steps $2-5$ until a stopping criterion is not satisfied (e.g. maximum number of iterations or a sufficiently good fitness value is not attained).

Our parallel PSO algorithm for object tracking consists of five main phases, namely initialization, evaluation, *p_best*, *g_best*, update and motion. At the beginning of each frame, in the initialization stage an initial position $x^{(i)} \leftarrow \mathcal{N}(p_{\mathrm{g}}, \Sigma)$ is assigned to each particle, given the location p_{g} that has been estimated in the previous frame. In the evaluation phase the fitness value of each

particle is calculated using a cost function. The calculation of the matching score is the most time consuming operation of the tracking algorithm. The calculation of the matching score is discussed in Section 4.2, whereas the decomposition of this task into kernels is presented in Section 4.3. In the *p_best* stage the determining of $p_{\text{best}}^{(i)}$ as well as $p^{(i)}$ takes place. This stage corresponds to operations from the point 3. of the presented above pseudo-code. The operations mentioned above are computed in parallel using available GPU resources, see Fig. 1. Afterwards, the g_{best} and its corresponding p_{g} are calculated in a sequential task. Finally, the update stage that corresponds to point 5. in the pseudo-code is done in parallel. That means that in our implementation we employ the parallel synchronous particle swarm optimization. The synchronous PSO algorithm updates all particle velocities and positions at the end of each optimization iteration. In contrast to synchronous PSO the asynchronous algorithm updates particle positions and velocities continuously using currently accessible information.

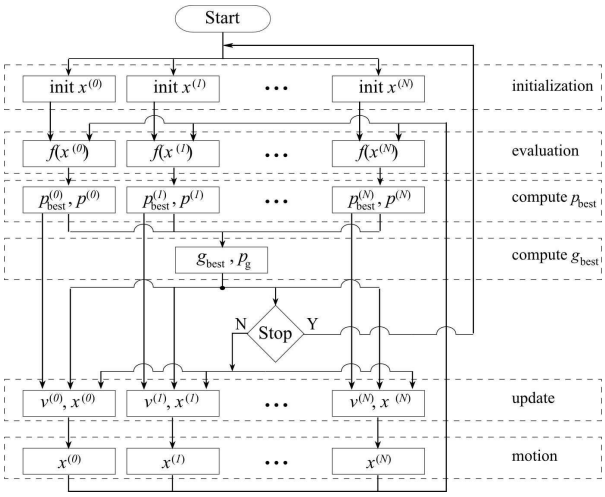

Fig. 1. Decomposition of synchronous particle swarm optimization algorithm on GPU.

In order to decompose an algorithm into 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$^{\text{TM}}$ SDK. On the basis of the uniform random numbers we generate normally distributed pseudorandom numbers using Box Mueller transform based on trigonometric functions [14]. The normally distributed random numbers are generated at the beginning of each frame to re-distribute the particles around the pose in time $t-1$ and to calculate their 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 $2 \times D \times K$ uniformly 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, see Fig. 1. 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 Implementation of Articulated Body Tracking on GPU

At the beginning of this section we detail our approach to 3D model based visual tracking of human motion. Afterwards, we present the cost function. Finally, we discuss the parallelization of the calculations of the cost function.

4.1 3D Model-Based Visual Tracking

The articulated model of the human body has a form of kinematic chain consisting of 11 segments. The 3D model is constructed using truncated cones (frustums) that model 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 A, B, C 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 angular 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. Though projecting all truncated cones we obtain the image representing the 3D model in a given configuration.

In each frame the 3D human pose is reconstructed through matching the projection of the human body model with the current image observations. 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 [15]. 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. The objective function of all cameras is the sum of such matching scores. Sample images from the utilized test sequences as well as details of camera setup can be found in [15].

The motion tracking can by attained by dynamic optimization and incorporating the temporal continuity information into the ordinary PSO. Consequently, it can be achieved by a sequence of static PSO-based optimizations, followed by re-diversification of the particles to cover the potential poses that can arise in the next time step. The re-diversification of the particle i can be obtained on the basis of normal distribution concentrated around the best particle location p_g in time $t-1$, which can be expressed as: $x^{(i)} \leftarrow \mathcal{N}(p_g, \Sigma)$, where $x^{(i)}$ stands for particle's location in time t, Σ denotes the covariance matrix of the Gaussian distribution, whose diagonal elements are proportional to the expected velocity.

4.2 Cost Function

The most computationally demanding operation in 3D model based human motion tracking is calculation of the objective function. In PSO-based approach each particle represents a hypothesis about possible person pose. In the evaluation of the particle's fitness score the projected model is matched with the current image observation. The fitness score depends on the amount of overlapping between the extracted silhouette in the current image and the projected and rasterized 3D model in the hypothesized pose. The amount of overlapping is calculated through checking the overlap degree from the silhouette to the rasterized model as well as from the rasterized model to the silhouette. The larger the 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 [15].

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

4.3 Parallelization of the Cost Function

In the evaluation phase, see Fig. 1 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. 2. 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. 2.

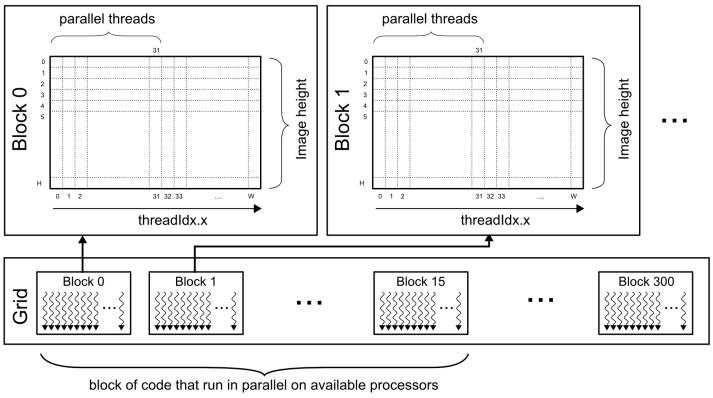

Fig. 2. Parallelization of the cost function.

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.

5 Experiments

The experiments were conducted 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 shows computation time that has been obtained on CPU and GPU for 1,2, and 4 cameras and PSO executing 10 iterations. For two cameras the computations were conducted on two CPU cores and two GPUs on single card, whereas for four cameras we employed 4 CPU cores and four GPUs. The images acquired from calibrated and synchronized cameras were preprocessed off-line and transferred frame by frame to the GPU. As we can observe, for a system

consisting of 2 cameras the speedup that achieves the GPU over the CPU is between 5.0 and 9.9. For 4 cameras the speed up is slightly smaller due to additional transmission overhead between two cards. For MoCap system consisting of 4 cameras and using the PSO algorithm with 300 particles and 10 iterations we can process 16 frames per second. In [15] we demonstrated that for such a PSO configuration the average error on images of size 960×540 is below 75 mm. In this work we employed the images scaled to 480×270 resolution and the average error was about 5 mm larger. Another reason for a slightly larger error is the use of synchronous PSO that achieves worse tracking accuracy in comparison to asynchronous PSO.

Table 1. Computation time [ms] for single frame of size 480×270.

	# part. (10 it.)	CPU [ms]	GPU [ms]	speedup
1 camera	100	131.1	24.6	5.3
	300	352.7	44.9	7.9
	1000	1134.7	106.4	10.7
2 cameras	100	132.8	26.8	5.0
	300	352.4	47.4	7.5
	1000	1117.4	113.5	9.9
4 cameras	100	170.3	37.3	4.6
	300	442.3	62.8	7.1
	1000	1391.9	144.7	9.6

The processing times on the CPU were obtained using an implementation presented in [15]. In [16] we showed that a modified PSO algorithm, i.e. annealed particle swarm optimization (APSO) [15], with 300 particles and executing 10 iterations, can be successfully used in 3D gait-based person identification.

6 Conclusions

In this paper we presented an algorithm for articulated human motion tracking on GPU. The tracking has been achieved in real-time using a parallel PSO algorithm. 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 speedup of the algorithm running on GPU over CPU grows with the number of evaluations of the cost function, i.e. with number of the particles or with the number of iterations. In consequence, on the GPU we can obtain more precise tracking.

Acknowledgment. This work has been supported by the National Science Center (NCN) within the research project N N516 483240.

References

1. Lee, V.W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A.D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., Dubey, P.: Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In: Proc. of the 37th Annual Int. Symp. on Computer Architecture. ISCA'10, New York, NY, USA, ACM (2010) 451–460
2. Castano-Diez, D., Moser, D., Schoenegger, A., Pruggnaller, S., Frangakis, A.S.: Performance evaluation of image processing algorithms on the GPU. Journal of Structural Biology **164**(1) (2008) 153 – 160
3. Pulli, K., Baksheev, A., Kornyakov, K., Eruhimov, V.: Real-time computer vision with OpenCV. Comm. ACM **55**(6) (June 2012) 61–69
4. Deutscher, J., Blake, A., Reid, I.: Articulated body motion capture by annealed particle filtering. In: IEEE Int. Conf. on Pattern Recognition. (2000) 126–133
5. Wu, C., Aghajan, H.K.: Human pose estimation in vision networks via distributed local processing and nonparametric belief propagation. In: Int. Conf. on Advanced Concepts for Intelligent Vision Systems, LNCS, Springer (2008) 1006–1017
6. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. of IEEE Int. Conf. on Neural Networks, IEEE Press, Piscataway, NJ (1995) 1942–1948
7. Krzeszowski, T., Kwolek, B., Wojciechowski, K.: GPU-accelerated tracking of the motion of 3d articulated figure. In: Proc. of the 2010 Int. Conf. on Computer Vision and Graphics: Part I, Berlin, Heidelberg, Springer-Verlag (Sept. 2010) 155–162
8. Mussi, L., Ivekovic, S., Cagnoni, S.: Markerless articulated human body tracking from multi-view video with GPU-PSO. In: Proc. of the 9th Int. Conf. on Evolvable Systems: from biology to hardware, Springer-Verlag (2010) 97–108
9. Zhou, Y., Tan, Y.: GPU-based parallel particle swarm optimization. In: IEEE Congress on Evolutionary Computation, CEC'09. (2009) 1493–1500
10. Laguna-Sanchez, G.A., Olguin-Carbajal, M., Cruz-Cortes, N., Barron-Fernandez, R., Alvarez-Cedillo, J.A.: Comparative study of parallel variants for a particle swarm optimization. J. of Applied Research and Technology **7**(3) (2009) 292–309
11. Solomon, S., Thulasiraman, P., Thulasiram, R.: Collaborative multi-swarm PSO for task matching using graphics processing units. In: Proc. of the 13th Annual Conf. on Genetic and Evolutionary Computation. (2011) 1563–1570
12. Brown, J., Capson, D.: A framework for 3d model-based visual tracking using a GPU-accelerated particle filter. IEEE Trans. on Visualization and Computer Graphics **18**(1) (2012) 68–80
13. Hansen, N., Auger, A., Ros, R., Finck, S., Pošík, P.: Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In: Genetic and Evolutionary Computation Conf. GECCO'10, ACM (2010) 1689–1696
14. Box, G.E.P., Muller, M.E.: A note on the generation of random normal deviates. The Annals of Mathematical Statistics **29**(2) (1958) 610–611
15. Kwolek, B., Krzeszowski, T., Wojciechowski, K.: Swarm intelligence based searching schemes for articulated 3D body motion tracking. In: Int. Conf. on Advanced Concepts for Intelligent Vision Systems. LNCS, vol. 6915, Springer (2011) 115–126
16. Krzeszowski, T., Michalczuk, A., Kwolek, B., Switonski, A., Josinski, H.: Gait recognition based on marker-less 3D motion capture. In: 10th IEEE Int. Conf. on Advanced Video and Signal Based Surveillance (AVSS). (2013) 232–237