Marker-Less 3D Human Motion Capture in Real-Time Using Particle Swarm Optimization with GPU-Accelerated Fitness Function

Bogdan Kwolek[1] and Boguslaw Rymut[2]

[1] AGH University of Science and Technology,
30 Mickiewicza Av., 30-059 Krakow, Poland, bkw@agh.edu.pl
[2] Rzeszow University of Technology, 12 Powst. Warszawy, 35-959 Rzeszów, Poland

Abstract. In model-based 3D motion tracking the most computationally demanding operation is evaluation of the objective function, which expresses similarity between the projected 3D model and image observations. In this work, marker-less tracking of full body has been realized in a multi-camera system using Particle Swarm Optimization. In order to accelerate the calculation of the fitness function the rendering of the 3D model in the requested poses has been realized using OpenGL. The experimental results show that the calculation of the fitness score with CUDA-OpenGL is up to 40 times faster in comparison to calculation it on a multi-core CPU using OpenGL-based model rendering. Thanks to CUDA-OpenGL acceleration of calculation of the fitness function the reconstruction of the full body motion can be achieved in real-time.

1 Introduction

Recovery of 3D human motion from images is an important problem in computer vision with many potential applications in areas such as interactive entertainment industry, sport or rehabilitation, interfaces for human-computer interaction, surveillance or augmented reality [14]. In general, the aim of 3D human motion recovery is to estimate the 3D joint locations of a human body from visual data. Marker-less motion tracking is one of the most challenging problems in computer vision being at the same time one of the most computationally demanding tasks. In recent years there has been a rapid progress in marker-less human pose recovery. Despite all these advances, 3D human motion reconstruction remains basically unsolved, particularly for unconstrained movement in dynamic and cluttered environments [16]. The challenge is not only to attain sufficient tracking accuracy, but also to develop solutions for real-time tracking [22,10].

The existing approaches for 3D motion reconstruction can either be described as part-based bottom-up approaches or model-based generative top-down methods. The appearance based approaches rely on image features and a regressor responsible for determining the corresponding pose within a set of predefined poses. Model based approaches define a parametric 3D model and search for the optimal solution in the model's continuous parameter space. The 3D pose reconstruction is achieved by local optimization around the previous pose estimate or by Bayesian filtering. Bottom–up approaches require big amount of training data to capture the large appearance variation that arises due to the highly articulated structure of the human pose. Top–down approaches provide a more accurate estimation of the human pose. Such approaches typically utilize a 3D articulated human body model. They render the body model and then compare the rendered images with the acquired images to calculate the fitness score. The key challenge is to cope with high-dimensionality of the search space as a result of large number of degrees of freedom (DOF). Hierarchical [22] or global-local approaches divide the searching for the best fitness into multiple stages, where a subset of the parameters is optimized, whereas the rest of them are fixed.

In recent years, many methods have been proposed for model-based motion tracking. Most work is based on particle filtering (PF) and its variants [2]. However, classical PF can be inefficient in high-dimensional state spaces since a large number of particles is needed to represent the posterior. More recently, stochastic global optimization methods such as Particle Swarm Optimization (PSO) have been gaining popularity in motion tracking [11] [14]. In PSO, unlike in PF, the particles interact with one another, and as they interact, a collective behavior arises, which leads to emergence of global and collective search capabilities allowing the particles to gravitate towards the global extremum.

To date, relatively little work has focused on accelerating 3D model based motion tracking, and particularly on accelerating the objective function being the most computationally intensive operation. As shown in [20], the computation time per frame of the most methods is from a few to several seconds. As recently demonstrated, considerable gains in number of frames per second in object tracking can be obtained through GPU-based accelerating the evaluation of the objective function or accelerating the evaluation of particle weights [9,4]. Indeed, thanks to substantial shortening of time needed for evaluation of the objective function on a GPU, and particularly accelerating the rendering that can account more than 75% of total tracking time [3], the 3D motion reconstruction can be done in real-time with a frequency of several frames per second [10,22].

This work is motivated by the need for acceleration of marker-less 3D motion reconstruction to achieve the tracking in real-time at high frame-rates. Our goal is to accelerate the 3D motion tracking through mixing graphics and compute. We present computation times of OpenGL-accelerated calculation of the fitness function on a multi-core CPU and a GPU. Computation times of the main ingredients of the fitness function are shown as well. We show that CUDA-OpenGL interoperability allows us to reconstruct the 3D motion of the full body in a four camera setup with more than 12 Hz. We demonstrate accuracy of the human motion tracking on freely available datasets.

2 CUDA-OpenGL Interoperability

General-purpose computation on the GPU (GPGPU) is a term denoting the employment of the GPU for calculations other than rendering. CUDA is scalable parallel programming model and a parallel programming platform for GPUs.

It enables efficient use of the massive parallelism of NVidia GPUs. The CUDA architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs). In this model, the host program launches a sequence of kernels, which are parallel portions of an application. A CUDA kernel is executed by an array of threads. All threads run the same code. The threads are organized into a grid of thread blocks. When a CUDA program on the host CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to multiprocessors. The threads of a thread block run concurrently on one multiprocessor, and multiple thread blocks can execute concurrently on one multiprocessor. As one thread blocks terminates, new blocks are launched on the available multiprocessors.

All threads within a single thread block are allowed to synchronize with each other via barriers and have access to a highspeed, per-block shared memory, which allows an inter–thread communication. Threads from different blocks in the same grid can coordinate only via operations in a global memory visible to all threads. In Kepler GK110 microarchitecture, each SMx contains 192 CUDA cores (CCs), and at any one time they can execute up to six wraps of 32 threads. GK110 also supports up to 64 resident warps per SM. In CUDA memory model, the global, constant, and texture memory spaces are persistent across kernel launches by the same application.

CUDA offers OpenGL interoperability features that allow to map OpenGL textures and buffers to be used inside a CUDA kernel without copying the whole content. The CUDA-OpenGL memory operations are described in [19].

3 Approach to Real-Time 3D Motion Tracking

The 3D model consists of truncated cones and each truncated cone is described by the center of top circle A, center of base circle B, top radius $r1$, and bottom radius $r2$. Figure 1 illustrates extraction of trapezoid representing the truncated cone in a given camera view. Having in disposal the 3D camera location C, 3D coordinates of the bottom circle center A and 3D coordinates of the top circle center B, and, we can determine the plane passing through the points A, B, C. Since the vectors AB and AC are in the same plane, their cross product being perpendicular to the plane, is the normal. The normal is used to calculate the angular orientation of the trapezoid. The projection of the trapezoid onto 2D image is performed on the basis of the camera model. The trapezoid image is generated by projecting the corners and then rasterizing the triangles composing the trapezoid. Though projecting every truncated cones composing of the 3D model we obtain the rendered image, which represents the 3D model in a given pose. In relevant literature the discussed technique is called billboarding [1].

In this work we also employed a more detailed 3D surface mesh model consisting of a surface representing the person skin and an underlying skeleton. Its surface has been deformed using a rigid skinning technique, in which the skeleton acts as a deformer transferring its motion to the skin, and where each skin vertex has assigned one joint as a driver.

The pose of the model is described by position and orientation of the pelvis in the global coordinate system and the relative angles between the limbs. The

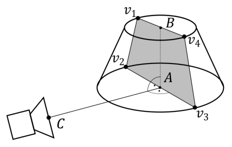


Fig. 1: Trapezoid representing truncated cone.

geometrical transformations of each body part are represented by a local transform matrix and a global transform matrix. The local transform matrixes describe geometrical transforms of model parts in their local coordinate systems, whereas the global transform matrixes represent geometrical transformations in the global coordinate system [13]. The beginnings of the local coordinate systems are in joints, whereas the beginning of the global coordinate system is the center of scene. Transformation of each body part is performed on the basis of rotation matrices describing global and local transformation of the body part [21,13]. The matrices are calculated on the basis of the joint Euler rotation angles stored in a state vector. In this paper, we formulate the 3D human pose tracking problem as a discrete optimization problem based on image matching in a calibrated multi-camera system and solve this problem using PSO [8]. Each particle of the swarm represents a hypothesized pose. The goal of the PSO is to find the best matching between the projected 3D model on the basis of the camera models and the images acquired by the calibrated and synchronized cameras. The evaluation of the system performance has been carried out on publicly available LeeWalk [2] and HumanEva I [17] image sequences. Both image sequences contain the parameters of the Tsai camera model [18] as well as ground-truth data. The LeeWalk image dataset consists of images of size 640×480 from four monochrome cameras, whereas HumanEva I dataset contains images of size 640×480 from 3 color cameras and images of the same size from 4 monochrome cameras. Both datasets were recorded with 60 fps.

Particle swarm optimization is a meta-heuristic that has been applied successfully to many optimization problems, including 3D human body pose tracking. In this meta-heuristic each individual is called particle and moves through a multidimensional space in order to find the best fitness score. The PSO algorithm seeks for a best fitness by iteratively trying to improve a candidate solution with regard to a given measure of quality. It optimizes a fitness function by maintaining a population of candidate solutions and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position, and is also guided toward the best known positions in the search-space. The best local and global positions are updated on the basis of the fitness function. The ordinary PSO algorithm can be used for human pose estimation in a single frame.

The 3D motion tracking over a sequence of frames can by attained by incorporating the temporal continuity information into the ordinary PSO. Thus, it can be achieved by a series of static PSO-based optimizations, followed by re-diversification of the particles to predict the body poses in the next frame.

The re-diversification of the particles can be achieved on the basis of normal distribution concentrated around the best pose found by the swarm.

Figure 2 illustrates how the synchronous PSO has been decomposed for a parallel execution on the GPU. The algorithm is executed in the six steps: swarm initialization, update of particle velocity, update of particle position, update of particle best position, update of swarm best position and fitness function evaluation. Each stage of the algorithm has been parallelized and then executed on the GPU. The most time consuming part of the motion tracking is rendering of the 3D model and matching of such a projected model with image observations.

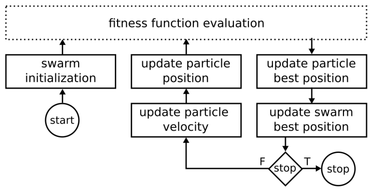


Fig. 2: Decomposition of synchronous PSO on GPU.

In this work, in order to speed-up the calculation of the fitness function the rendering of the 3D model is accelerated by OpenGL. The matching of the projected 3D models with the image observations is implemented in CUDA, whereas the transformation of the model to the requested poses as well as the rendering of the 3D model in the predicted poses is realized by OpenGL, see Fig. 3. The discussed framework is general and can be used in particle swarm optimization-based [14] or particle filtering-based [2] 3D pose inferring. The processing performance of CUDA-OpenGL based fitness function has been compared with performance of fitness function, which was determined on the basis of software CUDA-based rendering of the 3D model.

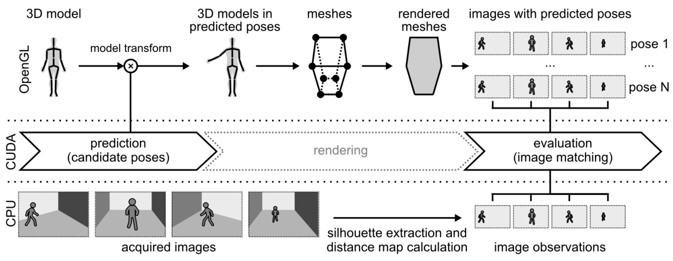

3D model
model transform
3D models in predicted poses
meshes
rendered meshes
images with predicted poses
pose 1
...
pose N
OpenGL
CUDA
prediction (candidate poses)
rendering
evaluation (image matching)
CPU
acquired images
silhouette extraction and distance map calculation
image observations

Fig. 3: Mixing graphics and compute to speed-up human body tracking.

4 Evaluation of the Fitness Function Using OpenGL API

At the beginning of this Section we present the objective function. Then, we discuss main steps in evaluation of the fitness score. Afterwards, we explain how OpenGL is used to accelerate the rendering of the 3D model. Then, we discuss the evaluation of the fitness score given the 3D models rendered by OpenGL.

4.1 Objective Function

The vision systems used in this work consist of two pairs of calibrated cameras, which are roughly oriented perpendicularly to each other. The objective function has been calculated in the following manner:

$$F(x) = \prod_{i=1}^{2} (f_i(x))^{w_i} \tag{1}$$

where w_i denotes a smoothing coefficient. The function $f_1(x)$ reflects degree of overlap between the projected 3D model into 2D images and the extracted silhouettes on the images acquired by the cameras. It has been calculated as:

$$f_1(x) = \beta \frac{\sum_{c=1}^{4} o_c}{\sum_{c=1}^{4} \hat{o}_c} + (1-\beta) \frac{\sum_{c=1}^{4} o_c}{\sum_{c=1}^{4} \hat{r}_c} \tag{2}$$

where o_c expresses the overlap degree between the silhouette of the projected 3D model and the silhouette extracted in the image from the camera c, $\hat{o}_c$ stands for the number of pixels belonging to the rendered silhouette for the camera c, $\hat{r}_c$ denotes the area of the silhouette extracted in the image from the camera c, whereas the β coefficient has been set to 0.5 to equally weight the overlap degree from the rasterized silhouette to the image silhouette and from the image silhouette to the rasterized silhouette. The function $f_2(x)$ expressing the distance map-based fitness between the edges has been calculated as follows:

$$f_2(x) = \frac{\sum_{c=1}^{4} d_c}{\sum_{c=1}^{4} e_c} \tag{3}$$

where e_c denotes the number of edge pixels in the rendered image for the camera c, whereas d_c reflects the sum of the distances from the edges extracted on the image from the camera c to the edges of the rendered model. The distances were calculated using the edge distance maps determined on the camera images.

The fitness score expressed by (1) is determined through matching the features on a pair of images. The first image contains the rendered silhouette and edges of the model, whereas the second one contains the silhouette and the distance to the edges, which have been extracted on the images acquired by the calibrated cameras. The first image is a subimage of the frame-buffer, which contains images corresponding to all particles. As aforementioned, information about the silhouette and the edge is stored in a single byte, where the first bit represents the presence of the silhouette, whereas the last one represents the occurrence of the edge of the projected model. In the second image the information is also stored in a single byte, where the first bit expresses the occurrence of the silhouette, whereas the seven remaining bits encode the distance of the pixels to the nearest edge. Given the extracted person silhouettes on the images acquired by the cameras a region of interest (ROI) is extracted. Each ROI is composed by enlarging the rectangle surrounding the extracted person about ten pixels both horizontally and vertically. Then, in order to permit 128-bit words transactions the left-upper coordinates of ROIs are shifted to be a multiple of four pixels. Finally, the width of every ROI is modified to be a multiple of four pixels.

4.2 Evaluation of Fitness Using Mixing Graphics and Compute

The evaluation of the fitness score is realized in three main steps:

- calculation of the global transformation matrixes
- 3D model rendering using OpenGL API
- Calculation of the fitness function value

In the first and third step, depending on the choice, CUDA or OpenGL or CPU is used to determine the global transformation matrixes needed by the OpenGL and to evaluate the fitness function given the rendered images by the OpenGL.

4.3 OpenGL-based Rendering of the 3D Model

The rendering result is stored as the color attachment in the frame-buffer consisting of 32-bits RGBA pixel values, where each component is represented by single byte. This means that in a single RGBA pixel we can store the pixel values of the rendered model onto images of four cameras or the values of four models in different hypothetical poses. Having on regard that in the second case the rendered models are rather in similar poses in comparison to pose estimated in the last frame, we selected this method to store the all rendered models. In consequence, the rendered models, which are stored in four components of an RGBA image, are of similar size and shape.

In the rendering of the 3D model, three programmable pipelines were executed: (i) vertices calculation, (ii) silhouette rendering and (iii) outline (edge) rendering. The pipelines were programmed by shader programs written in the GLSL language.

The Vertex calculation pipeline uses geometry instancing [15] for creating instances of model vertices. The number of created instances is equal to number of particles multiplied by number of cameras used in the tracking. Each generated model instance is transformed using the world/global transformation matrices of a corresponding particle, which is stored in SSBO [15]. After calculating the model transformation, all vertices of an instance are projected into image coordinates using Tsai camera model. To render more than one model image on a frame-buffer each vertice of the model instance is shifted in different frame-buffer region, using a method commonly known as image tiling [12]. Finally the computed vertices are stored in vertex feedback buffer object [15]. The vertices stored in this buffer are then used in the silhouette rendering and edge rendering pipelines. Vertex calculation pipeline uses vertex shader program and geometry shader program for creating instances, projecting model onto image and tiling model images (subimages) into frame-buffer.

The silhouette rendering pipeline uses vertices stored in the vertex feedback buffer object (VFBO) to render particle models using triangles primitive on a frame-buffer. In this pipeline the element array buffer object (EABO) containing indexes of vertices stored in the VFBO is used to pull vertices in proper order and to render model appearance using triangles primitive. Silhouette rendering pipeline uses dummy vertex shader program and fragment shader program to pass the input vertices to hardware rasterization stage.

Outline (edge) rendering pipeline is almost identical to silhouette rendering pipeline. The only notable difference is the use of adjusted triangle primitive as an input and geometry shader program. Geometry shader program processes adjusted each triangle primitive to detect edge between adjusted triangles by simplified streamlined method [6]. The geometry shader emits new line primitive when an edge between two triangles is detected. The emitted line primitive vertices are then passed to the next rendering stage and drawn on frame-buffer. Hence, the number of the rendered subimages is equal to the number of the particles and the size of each subimage is equal to the size of the camera image.

The painting of the silhouettes is realized using color blending. The information about the silhouette and the edge is stored in a single byte, where the first bit represents the presence of the silhouette, whereas the last one represents the occurrence of the edge of the projected model. After the OpenGL synchronization, the frame-buffer is mapped to CUDA memory and then used by CUDA in computation of the fitness score.

4.4 Fitness Evaluation Given the 3D Models Rendered by OpenGL

The value of the fitness score (1) is determined in two kernels, where the first one calculates components o_c, $\hat{o}_c$ of function (2) and components d_c, e_c of function (3), whereas the second kernel uses the values of the components to determine the value of function (1). The main computational load is connected with calculation of the components o_c, $\hat{o}_c$, d_c and e_c. Moreover, the size of ROIs changes as the person undergoing tracking moves on the scene, which in turn can lead to unequal computational burden. The first kernel is far more computationally demanding than the second one.

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 NVidia GTX 780 Ti graphics card consisting of 15 multiprocessors and 192 cores per multiprocessor. The card is equipped with 3072 MB RAM and 64 KB on-chip memory per multiprocessor. In the Kepler GK110 architecture, the on-chip memory can be configured as 48 KB of shared memory with 16 KB of L1 cache, or as 16 KB of shared memory with 48 KB of L1 cache.

Table 1 presents execution times needed for determining the objective function values in the single iteration of PSO. The presented times were obtained by averaging times for determining the objective functions values during 3D pose estimation in all 450 frames of the LeeWalk sequence. In the discussed test the PSO executed 10 iterations and consisted of 96, 304 and 512 particles, respectively. As we already mentioned, the images in this four-camera test sequence are 480 pixels high and 640 pixels wide.

The bottom part of Tab. 1 presents the overall processing time together with the compute time. The software rendering algorithms, which are responsible

Table 1: Execution times [ms] of the main components of the fitness function.

	#particles	CPU [ms]	CUDA [ms]	CPU-OpenGL [ms]	CPU SSE-OpenGL [ms]	CUDA-OpenGL [ms]
World transformation matrixes	96	0.21	0.85	0.21	0.17	0.83
	304	0.52	0.92	0.52	0.45	0.77
	512	0.91	0.93	0.91	0.76	0.82
Sending world transf. matrixes to OpenGL	96	-	-	0.26	0.25	0.26
	304	-	-	0.31	0.30	0.28
	512	-	-	0.34	0.31	0.30
OpenGL rendering	96	-	-	1.48	1.48	1.40
	304	-	-	2.81	2.74	2.71
	512	-	-	3.92	3.89	3.81
Acquire rendering result from OpenGL	96	-	-	63.37	62.88	-
	304	-	-	201.47	212.91	-
	512	-	-	309.53	318.27	-
Mapping rendering result (memory mapping)	96	-	-	-	-	0.40
	304	-	-	-	-	0.34
	512	-	-	-	-	0.38
Evaluate fitness value	96	26.13	13.87	14.86	6.72	1.50
	304	92.03	22.44	50.06	18.96	2.22
	512	176.32	36.98	79.23	30.40	2.92
Overall time	96	26.34	14.72	80.18	71.50	4.40
	304	92.55	23.36	255.17	235.36	6.30
	512	177.23	37.91	393.93	353.63	8.23
Compute time (without acquisition, mapping and transmission)	96	26.34	14.72	16.55	8.37	3.73
	304	92.55	23.36	53.39	22.15	5.70
	512	177.23	37.91	84.06	35.05	7.55

for rendering of triangles and edges, perform point inclusion tests for simple polygons [5]. The visible surfaces are determined using painter's algorithm [7]. The processing on the CPU has been realized on four cores using OpenMP. As we can notice, the processing times obtained on the GPU are far shorter than processing times achieved on the CPU. As we can observe, the compute time on the CPU can be considerably reduced through the use of the SSE instructions. By comparing the overall times and compute times presented in the table, we can observe that CPU-OpenGL and CPU SSE-OpenGL have considerable overheads for data acquisition, mapping and transmission. Having on regard that the 3D model rendering is the most time consuming operation in model-based 3D motion reconstruction, the use of OpenGL leads to far more larger number of processed frames per second (fps). Thanks to effective utilization of the rendering power of OpenGL to render the 3D models in the predicted poses the motion tracking can be done in real-time at high frame-rates.

The use of OpenGL-based 3D model rendering makes shorter the tracking time in comparison to software (CUDA) rendering, see also Fig. 4. It is also worth noting that for the small number of the particles, say up to 300, the tracking time of the algorithm with OpenGL-based rendering does not change considerably with the growing particle population.

The fitness functions discussed above were utilized in a PSO-based algorithm for marker-less motion tracking. Its accuracy has been evaluated on the

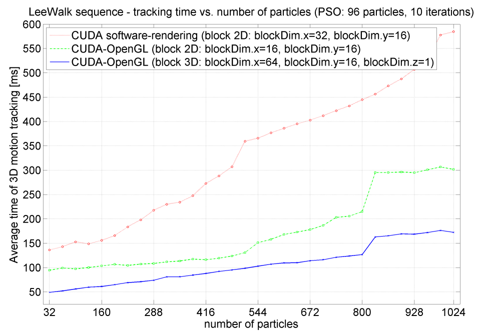


Fig. 4: 3D Motion tracking time for OpenGL and soft (CUDA) rendering.

mentioned above LeeWalk test sequence. Figure 5 illustrates the tracking errors versus frame number. The tracking accuracy on LeeWalk image sequence is comparable to accuracy reported in [22].

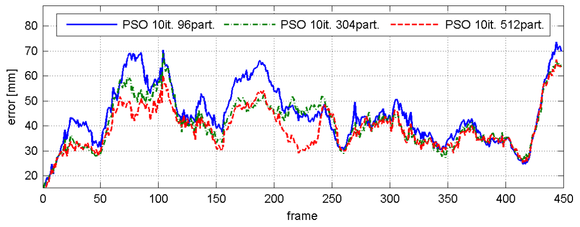
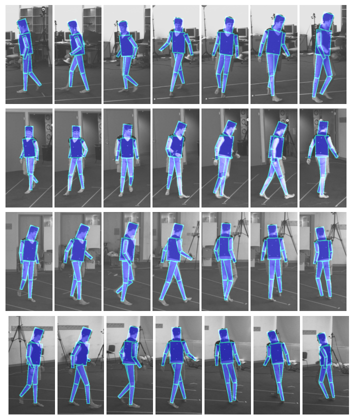


Fig. 5: Tracking error [mm] vs. frame number on LeeWalk dataset.

Figure 6 illustrates the tracking accuracy that was obtained on LeeWalk image sequence. The larger the overlap between the projected model and the person silhouette is, the better is the 3D humane pose estimate.



#20 40 60 80 100 120 140

Fig. 6: 3D human body tracking on LeeWalk sequence.

Table 2 presents the tracking accuracies, which were achieved on LeeWalk and HumanEva I image sequences. The above mentioned results were achieved in ten independent runs of the PSO-based motion tracker with unlike initializations.

Table 2: Avarage tracking errors [mm] on LeeWalk and HumaEva I.

	Average tracking error using PSO (10 it.)		
	96 part.	304 part.	512 part.
LeeWalk	47.9 ± 12.5	43.3 ± 10.3	40.0 ± 11.7
HumanEva I	55.8 ± 21.3	53.5 ± 18.4	52.3 ± 19.1

The results presented in Tab. 3 show the average times that are needed to estimate the pose in a single frame using the PSO. It presents the average times that were obtained on LeeWalk image sequence. As we can observe, the method relying on truncated cones and trapezoid has shorter times in comparison to mesh-based method. Due to noisy images the mesh-based method did not give noticeably better tracking accuracies.

Table 3: Tracking times [ms] for different 3D models.

		Number of particles	
Model type	#vert.	96	304
trapezoid	60	75.7 ± 5.9	90.9 ± 7.2
mesh	250	79.1 ± 7.2	103.5 ± 5.3
	500	86.7 ± 7.0	114.3 ± 4.9
	1000	90.7 ± 6.0	128.8 ± 5.0
	4000	131.6 ± 7.6	237.4 ± 5.8

6 Conclusions

In this work we demonstrated a framework for marker-less 3D human motion tracking in real-time using PSO with GPU-accelerated fitness function. We demonstrated that OpenGL-based rendering of the 3D model in marker-less human motion tracking allows us to considerably shorten the computation time of the objective function, which is the most computationally demanding operation. We evaluated the interoperability between CUDA and OpenGL, which permitted mixed compute and rendering acceleration. Thanks to such an acceleration the full-body tracking can be realized in real-time.

Acknowledgment. This work was supported by Polish Ministry of Science and Higher Education under grant No. U-711/DS (B. Rymut) and Polish National Science Center (NCN) under research grant 2014/15/B/ST6/02808 (B. Kwolek).

References

1. Akenine-Möller, T., Haines, E., Hoffman, N.: Real-Time Rendering, 3rd Edition. A. Peters, Ltd., Natick, USA (2008)

2. Balan, A.O., Sigal, L., Black, M.J.: A quantitative evaluation of video-based 3D person tracking. In: Int. Conf. on Comp. Comm. and Networks. pp. 349–356 (2005)
3. Cano, A., Yeguas-Bolivar, E., Munoz-Salinas, R., Medina-Carnicer, R., Ventura, S.: Parallelization strategies for markerless human motion capture. Journal of Real-Time Image Processing (2015)
4. Concha, D., Cabido, R., Pantrigo, J.J., Montemayor, A.: Performance evaluation of a 3D multi-view-based particle filter for visual object tracking using GPUs and multicore CPUs. Journal of Real-Time Image Processing (2015)
5. Feito, F., Torres, J.C., Urea-López, L.A.: Orientation, simplicity and inclusion test for planar polygons. Computers & Graphics 19(4), 596–600 (1995)
6. Hajagos, B., Szécsi, L., Csébfalvi, B.: Fast silhouette and crease edge synthesis with geometry shaders. In: Proc. Spring Conf. on Comp. Graphics. pp. 71–76 (2012)
7. Hughes, J., Van Dam, A., McGuire, M., Sklar, D., Foley, J., Feiner, S., Akeley, K.: Computer Graphics: Principles and Practice. Addison-Wesley (2013)
8. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. of IEEE Int. Conf. on Neural Networks. pp. 1942–1948. IEEE Press, Piscataway, NJ (1995)
9. Krzeszowski, T., Kwolek, B., Wojciechowski, K.: GPU-accelerated tracking of the motion of 3D articulated figure. In: Proc. ICCVG. pp. 155–162. LNCS, vol. 6374, Springer (2010)
10. Kwolek, B., Krzeszowski, T., Gagalowicz, A., Wojciechowski, K., Josinski, H.: Real-time multi-view human motion tracking using particle swarm optimization with resampling. In: Int. Conf. AMDO, LNCS, vol. 7378, pp. 92–101. Springer (2012)
11. Kwolek, B., Krzeszowski, T., Wojciechowski, K.: Swarm intelligence based searching schemes for articulated 3d body motion tracking. In: Proc. 13th Int. Conf. on Adv. Concepts for Intell. Vision Syst. pp. 115–126. LNCS, vol. 6915 (2011)
12. McReynolds, T., Blythe, D.: Advanced Graphics Programming Using OpenGL. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
13. Parent, R.: Advanced algorithms. In: Parent, R. (ed.) Computer Animation, pp. 173 – 270. Morgan Kaufmann, San Francisco (2002)
14. Saini, S., Rambli, D.R., Zakaria, M., Sulaiman, S.: A review on particle swarm optimization algorithm and its variants to human motion tracking. Mathematical Problems in Engineering (2014)
15. Segal, M., Akeley, K.: The OpenGL Graphics System. A Specification, Version 4.3. Khronos Group (2013)
16. Shaheen, M., Gall, J., Strzodka, R., Van Gool, L., Seidel, H.P.: A comparison of 3D model-based tracking approaches for human motion capture in uncontrolled environments. In: Workshop on Appl. of Computer Vision (WACV). pp. 1–8 (2009)
17. Sigal, L., Black, M.J.: HumanEva: Synchronized video and motion capture dataset for evaluation of articulated human motion. Tech. Rep. CS-06-08, Brown University, Department of Computer Science (2006)
18. Song, L., Wu, W., Guo, J., Li, X.: Survey on camera calibration technique. In: Int. Conf. on Intel. Human-Machine Systems and Cyb. vol. 2, pp. 389–392 (2013)
19. Stam, J.: What every CUDA programmer should know about OpenGL. In: GPU Technology Conference (2009)
20. Yao, A., Gall, J., Gool, L.: Coupled action recognition and pose estimation from multiple views. Int. J. Comput. Vision 100(1), 16–37 (Oct 2012)
21. Zatsiorsky, V.: Kinematics of Human Motion. Human Kinetics (1998)
22. Zhang, Z., Seah, H.S., Quah, C.K., Sun, J.: GPU-accelerated real-time tracking of full-body motion with multi-layer search. Multimedia, IEEE Trans. on 15(1), 106–119 (Jan 2013)