Markerless Articulated Human Body Tracking from Multi-View Video with GPU-PSO

Luca Mussi[1], Spela Ivekovic[1,2], and Stefano Cagnoni[1]

[1] Dept. of Information Engineering, University of Parma, Italy
[2] Lessells Scholar, Royal Society of Edinburgh, Scotland

Abstract. In this paper, we describe the GPU implementation of a markerless full-body articulated human motion tracking system from multi-view video sequences acquired in a studio environment. The tracking is formulated as a multi-dimensional nonlinear optimisation problem solved using particle swarm optimisation (PSO). We model the human body pose with a skeleton-driven subdivision-surface human body model. The optimisation looks for the best match between the silhouettes generated by the projection of the model in a candidate pose and the silhouettes extracted from the original video sequence. In formulating the solution, we exploit the inherent parallel nature of PSO to formulate a GPU-PSO, implemented within the nVIDIA™ CUDA™ architecture. Results demonstrate that the GPU-PSO implementation recovers the articulated body pose from 10-viewpoint video sequences with significant computational savings when compared to the sequential implementation, thereby increasing the practical potential of our markerless pose estimation approach.

1 Introduction

Articulated human body pose estimation is an active research area with solutions applicable in many domains, including virtual character animation, biometrics, human-computer interaction, gait analysis, video surveillance, and others. While most industrial solutions still tend to rely on marker-based systems, such as Vicon [23], the advances in the markerless video-based estimation are progressing rapidly [16]. The attraction of the markerless pose estimation lies in the reduced preparation time for each capture session as well as the non-invasive nature of the procedure. In markerless capture, the use of tight body suits and magnetic or optical markers is not necessary; instead, the subjects can normally take part in their every-day clothing. Replacing the marker-based systems with markerless solutions, such as the one described in this paper, opens the possibility of using motion capture in areas such as medical analysis and home entertainment, where the use of tight body suits and markers is not acceptable. Additionally, the increasing availability and affordability of the video cameras makes markerless motion capture an ever more attractive alternative.

Modelling the articulated structure of the full human body for the purpose of pose estimation requires a large number of parameters, typically at least 30. The articulated pose estimation problem is therefore usually formulated as a search in a high-dimensional parameter space, which is invariably computationally very complex. In this paper, we address the issue of complexity by exploring the parallel nature of the

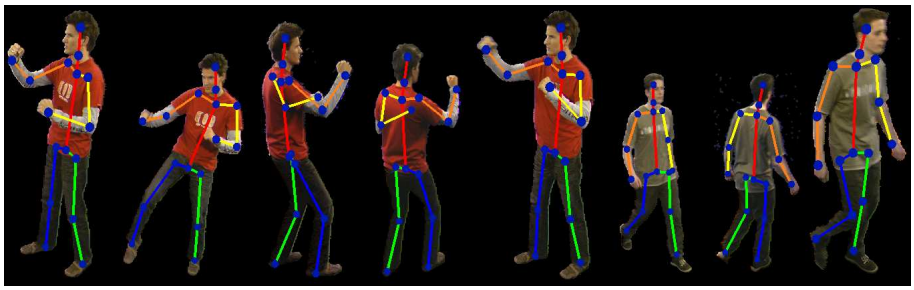

Fig. 1. Example pose results shown as skeletons overlaid on the corresponding input image. The examples shown are taken from different sequences (Jon Walk, Tony Kick, Tony Punch and Tony Stance) and different camera views (10 views were used for each sequence), hence the difference in person size as well as orientation.

markerless pose estimation problem at hand and searching the corresponding large parameter space using PSO. We exploit the fact that the PSO solution naturally lends itself to a parallel implementation on the state-of-the-art CUDA™ architecture, as well as that the multi-view pose estimation, based on silhouette comparison, itself contains a degree of parallelism that can be exploited to design a more efficient solution.

This paper is organised as follows. We begin with an overview of the related work in Section 2. In Section 3 we outline the CUDA™ architecture and present the PSO algorithm developed for it. Our pose estimation algorithm is presented in Section 4. Finally, we report experimental results in Section 5 and conclude with Section 6.

2 Related Work

In this section, we review the related work relevant to our approach. We begin with the related research in articulated human body pose estimation and then review the basics of PSO and relevant research in the area of PSO parallelisation.

2.1 Articulated human body pose estimation from video

Articulated 3-D human body pose estimation from video is an active research area [13, 19]. The complexity of the human body pose parametrisation has invariably required the pose estimation to be formulated as a high-dimensional space search problem and research has focused on reducing the complexity of the search. Various implementations of particle filters quickly gained popularity [1, 4]. Partitioning the search space into smaller, more manageable subspaces is also a popular approach [1, 12]. Furthermore, given the complexity of the articulated human body motion, the standard motion models used in the tracking literature did not suffice and attempts were made to learn motion models for particular actions from training data collected in advance [2, 21].

The above mentioned approaches suffer from various setbacks. The particle-filtering solutions critically rely on a high number of particles to adequately represent the poste-

rior distribution, which in turn increases their computational complexity beyond practical use when considering a wide variety of motion. The tendency to rely on pre-trained motion models causes the human body tracking approaches to lose their generalisation abilities. In order to address that, researchers started turning to methods which could reliably provide motion estimates without a pre-trained motion model [5]. In this paper, we explore a similar direction. We use a powerful search algorithm which is capable of recovering the pose without any prior knowledge of the nature of motion. The main advantage of such an approach is in its generality as it can estimate any kind of body motion when provided with a sufficient number of constraints, in our case image silhouettes. The downside is that it requires a lot of computation time. Previous attempts at reducing the computational complexity have focused on algorithmic improvements [7–9]. However, as we show in this paper, exploiting the parallel nature of both the search algorithm and the multi-view pose estimation problem by implementing the approach on a graphical processing unit (GPU) provides a natural alternative solution which is significantly more efficient while being equally general.

2.2 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) [10] is a powerful optimisation algorithm which searches the optimum of a fitness function following rules inspired by the behaviour of flocks of birds looking for food. As a population based meta-heuristic, PSO has recently gained popularity due to its robustness, effectiveness, and simplicity.

A particle's position and velocity within the domain of the fitness function at time t can be computed using the following equations:

$$\begin{aligned} \boldsymbol{V}(t) = w\ \boldsymbol{V}(t-1)\ + \\ C_1\ R_1\ [\boldsymbol{X}_{best}\ (t-1) - \boldsymbol{X}(t-1)]\ + \\ C_2\ R_2\ [\boldsymbol{X}_{gbest}(t-1) - \boldsymbol{X}(t-1)] \end{aligned} \tag{1}$$

$$\boldsymbol{X}(t) = \boldsymbol{X}(t-1) + \boldsymbol{V}(t) \tag{2}$$

where $\boldsymbol{V}$ is the velocity of the particle, C_1, C_2 are positive constants, R_1, R_2 are random numbers uniformly drawn between 0 and 1, w is the so-called 'inertia weight', $\boldsymbol{X}(t)$ is the position of the particle at time t, $\boldsymbol{X}_{best}(t-1)$ and $\boldsymbol{X}_{gbest}(t-1)$ are, respectively, the best-fitness position reached by the particle and the best-fitness point ever found by the whole swarm up to time $t-1$.

Many variants of the basic algorithm have been developed [18], some of which define different topologies for particles' neighbourhoods. A usual variant of PSO substitutes $\boldsymbol{X}_{gbest}(t-1)$ with $\boldsymbol{X}_{lbest}(t-1)$, the best position ever found within a pre-set neighbourhood of the particle under consideration. This formulation admits, in turn, several variants, depending on the neighbourhood topology.

Another factor that affects the performance of PSO is the order by which $\boldsymbol{X}_{gbest}$ / $\boldsymbol{X}_{lbest}$ are updated. In 'synchronous' PSO, during each iteration, positions and velocities of all particles are updated one after another in turn, after which each particle's fitness is evaluated. Finally, when the fitness of all particles is known, the value of $\boldsymbol{X}_{gbest}$ / $\boldsymbol{X}_{lbest}$ is updated. The 'asynchronous' version of PSO, instead, updates

$\boldsymbol{X}_{gbest}$ / $\boldsymbol{X}_{lbest}$ immediately after evaluation of each particle's fitness, leading to a more 'reactive' swarm which is attracted more promptly by newly-found optima.

Despite good convergence properties, PSO is still an iterative process which may require millions of particle updates and fitness evaluations. This makes the design of efficient PSO implementations a problem of great practical relevance, especially for real-time applications to dynamic environments. This is the case, for example, of computer vision applications in which PSO has been used to determine location and orientation of objects [14, 15] or posture of people [8].

PSO parallelisation has therefore become a popular subject for research. Before GPU-based programming environments were available, PSO was implemented following more traditional parallel computing paradigms, as in [6, 20]. Some of the implementations were hybridised with evolutionary algorithm paradigms, such as the so-called 'island model', obtaining a coarse-grained parallelisation [3, 24]. Conversely, research on fine-grained parallel PSO algorithms has mainly focused on the swarm topology. One of the first GPU-based PSO implementations was based on a fine-grained approach [11] which, however, was still based on 'hand-coded' texture-rendering mapping and did not rely on any GPU-specific programming environment. An overview of published work according to granularity analysis can be found in [24].

An interesting classification of parallel PSO algorithms, based on the best position update strategy, is reported in [27].

The most recent implementations are GPU-based [22, 25, 28], mostly developed within the CUDA™ environment, like the parallel PSO algorithm which we have developed and used in this work. Comparisons on the same benchmarks (not yet published) suggest that our approach outperforms these in terms of computation efficiency.

3 Parallel PSO Implementation Within the CUDA™ Architecture

CUDA™ (Compute Unified Distributed Architecture) is a parallel computing environment by nVIDIA™ which exploits the massively parallel computation capabilities of its most recent GPUs. CUDA™'s programming model requires that the problem under consideration be partitioned into many independent sub-tasks (*thread blocks*) which are solved in parallel by a number of cooperating *threads*. In the CUDA™ abstraction a programmer can define a two-dimensional grid of thread blocks; each block is associated with a unique pair of indices that identifies it within the grid. Within each block, as well, the threads that compose it can be organised as a two- or three-dimensional grid within which they are identified by a unique set of indices. This mechanism allows each thread to personalise its access to data structures and to decompose problems effectively.

From a hardware viewpoint, a CUDA™-compatible GPU is made up of a scalable array of multithreaded Streaming Multiprocessors (SMs), each of which is able to execute several thread blocks at the same time. Each SM embeds eight scalar processing cores and is equipped with a number of fast 32-bit registers, a parallel data cache shared among all cores, a read-only constant cache and a read-only texture cache accessed via a texture unit that provides several different addressing/filtering modes. In addition, SMs can access local and global memory spaces which are (non-cached) read/write regions

Listing 1.1. Synchronous PSO pseudo-code

```
<Initialise positions/velocities of all particles>
<Perform a first evaluation of the fitness functions>
<Set initial personal/global bests>
for ( i = 0; i < generationsNumber; i++){
    <Update the position of all particles>
    <Re−evaluate the fitness of all particles>
    <Update all personal/global bests>}
<Retrieve global best information to be returned as final result>
```

of device memory: these memories are characterised by latency times about two orders of magnitude larger than the registers and texture cache. Only threads belonging to the same thread block can share data in fast memory; different thread blocks may only share data allocated in slow memory. CUDA™'s scheduler allocates as many thread blocks at the same time as possible, compatibly with available resources, which permits a CUDA™ program to be run on any number of SMs. SMs can manage hundreds of threads running different code segments thanks to an architecture called SIMT (Single Instruction, Multiple Thread) which creates, manages, schedules, and executes groups (warps) of 32 parallel threads. Opposite to what happens in a SIMD (Single Instruction, Multiple Data) architecture, the whole execution and branching behavior of threads is specified. This way it is possible to manage parallel code for independent scalar threads as well as code for parallel data processing, which is executed by coordinated threads[3,4].

3.1 Parallelising PSO using CUDA™

The structure of PSO is very close to being intrinsically parallel. In PSO, the only dependence between the processes which update the particles' velocities and positions is related to the information which must be shared among the particles. This information is either only $\boldsymbol{X}_{gbest}$ or the corresponding vector $\boldsymbol{X}_{lbest}$ of the best positions found by any member of each particle's neighbourhood.

The most natural way to remove the dependence between particles' updates would consist of implementing synchronous PSO, updating $\boldsymbol{X}_{gbest}$ or $\boldsymbol{X}_{lbest}$ only at the end of each iteration. While this would permit the use of a single thread block (with one thread per particle) to implement a swarm, while avoiding accesses to global memory, it would impose limitations to the implementation of the fitness function and use computing resources inefficiently.

To better exploit the capabilities offered by CUDA™ in developing a parallel PSO algorithm, we considered the main stages of the algorithm as separate tasks, which can be parallelised differently. Listing 1.1 shows the pseudo-code of a synchronous PSO algorithm, regardless of implementation. In our case the three stages of the main loop are implemented as different kernels sequentially scheduled by the GPU. This does not affect execution time since kernel scheduling is very efficient. However, it imposes that

[3] nVIDIA CUDA C programming - Best practices guide v. 2.3, nVidia Corporation, May 2010
[4] nVIDIA CUDA programming guide v. 2.3, NVidia Corporation, May 2010

each kernel must load all data it needs initially and store it back at the end of every execution, since in CUDA™ data can be shared among kernels only through the (slow) global memory. Despite this, having limited the number of such accesses and organised data in order to exploit the GPU coalescing capability, the multi-kernel approach turned out to be more efficient. The first kernel (*PositionUpdateKernel*) updates the particles' positions scheduling a number of thread blocks equal to the number of particles; each block updates the position of one particle running a number of threads equal to the problem dimension D. The second kernel (*FitnessKernel*) is used to compute the fitness. Depending on the fitness function structure, i.e., its parallel nature, more than one kernel can be used at this stage to maximise the use of GPU resources. The last kernel (*BestUpdateKernel*) updates $\boldsymbol{X}_{gbest}$ and $\boldsymbol{X}_{lbest}$. Since its structure must reflect the swarm topology, the number of thread blocks to be scheduled may vary from one per swarm, in case of global-best topology, to many per swarm (to have one thread per particle), in case of ring topology.

Pseudo-random numbers are directly generated on the GPU using the Mersenne Twister kernel available in the CUDA™ SDK. Based on the available amount of device memory, we run this kernel every given number of PSO iterations. Pseudo-random numbers are stored in a dedicated array which can be accessed by other kernels.

4 Pose Estimation Algorithm

In this section we provide a detailed description of the articulated pose estimation problem and its building blocks. We describe the articulated human body model which we use to represent the candidate body poses, formulate the pose estimation as a PSO-search and define the cost function used to evaluate the quality of a candidate pose.

4.1 Body Model

To represent the candidate body pose, we use a 3-D layered subdivision surface body model consisting of two layers, the *skeleton* and the *skin*. The skeleton layer is defined as a set of homogeneous 4×4 transformation matrices T_i which encode the information about the position and orientation of every joint with respect to its parent joint in the kinematic tree hierarchy:

$$Skeleton = \{T_0, T_1, T_2, ..., T_{20}\}, \tag{3}$$

where T_i, $i = 0 \dots 20$, is a homogeneous transformation matrix encoding the orientation of the coordinate system of joint i with respect to the coordinate system of the preceding joint, specified by the kinematic tree shown in Figure 2.

The skin layer, which represents the second layer in the model, is connected to the skeleton through the joints' local coordinate systems. Each joint controls a certain area of the skin. Whenever a joint or limb moves, the corresponding part of the skin moves and deforms with it. As the skin is a subdivision surface, only the base mesh has to be specified in the corresponding joint's coordinate system. After the joint's configuration has been specified, the base mesh is subdivided by repeatedly applying the Catmull-Clark subdivision operator until the desired smooth shape of the body is obtained [26].

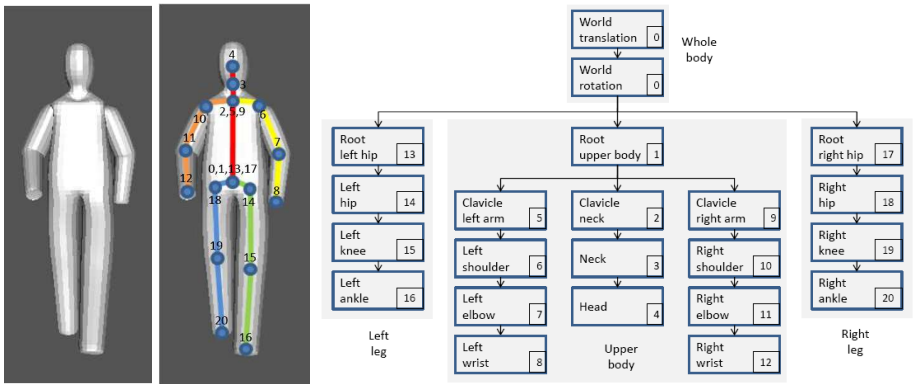

Fig. 2. Catmull-Clark subdivision surface body model and the corresponding skeletal hierarchy. In the full hierarchy, every joint has 3 rotational and 3 translational degrees of freedom (DOF). For the purpose of our work, we choose a subset of rotational DOF, detailed in Table 1. We also fix the limb lengths and only optimise the global position of the body in space.

4.2 PSO parametrisation of the articulated pose

In PSO, each particle represents a potential solution in the search space. Our search space is the space of all plausible skeleton configurations. The individual particle's position vector in the search space is specified as follows:

$$\boldsymbol{X}_i = (r_x, r_y, r_z, \alpha_x^0, \beta_y^0, \gamma_z^0, \alpha_x^1, \beta_y^1, \gamma_z^1, ..., \gamma_z^M), \tag{4}$$

where i denotes the index of the particle in the swarm, r_x, r_y, r_z denote the position of the root joint with respect to the reference (world) coordinate system, and $\alpha_x^j, \beta_y^j, \gamma_z^j$ refer to rotational degrees of freedom of joint j around the x, y, and z-axis, respectively. The total number of joints (the root joint has both translational and rotational degrees of freedom) is $M + 1$. As not all joints that are used to display the body need to be optimised, the joints and their respective degrees of freedom actually used in our pose estimation algorithm are given in Table 1.

4.3 Search Hierarchy

Searching for the correct articulated pose configuration in a 32-dimensional search space is expensive. Fortunately, the hierarchy in the kinematic structure of the human body allows for the search to be formulated as a sequence of steps in which only a subset of the 32 parameters is optimised at any one time. The hierarchy has the form of a kinematic tree and is illustrated in Figure 2. We formulate the search algorithm as 11 disjoint steps (equivalent to splitting the 32-dimensional search space into 11 disjoint subspaces) detailed in Table 2, where the solution of each step constrains the search space for the steps which follow. The individual steps are chosen so that only one limb segment at a time is optimised.

Table 1. Joints used to describe the configuration of the human body pose and their respective degrees of freedom used in the pose estimation algorithm. There are 32 DOF in total. The numbers in parentheses refer to the transformations in Figure 2

JOINT (index)	#	DOF	JOINT (index)	#	DOF
Global body position (0)	3	r_x, r_y, r_z	Right elbow orientation (11)	1	γ_z^{11}
Torso orientation (1)	3	$\alpha_x^1, \beta_y^1, \gamma_z^1$	Root left hip orientation (13)	2	$\alpha_x^{13}, \gamma_z^{13}$
Head orientation (2)	2	α_x^2, γ_z^2	Left hip orientation (14)	3	$\alpha_x^{14}, \beta_y^{14}, \gamma_z^{14}$
Left clavicle orientation (5)	2	α_x^5, γ_z^5	Left knee orientation (15)	1	γ_z^{15}
Left shoulder orientation (6)	3	$\alpha_x^6, \beta_y^6, \gamma_z^6$	Root right hip orientation (17)	2	$\alpha_x^{17}, \gamma_z^{17}$
Left elbow orientation (7)	1	γ_z^7	Right hip orientation (18)	3	$\alpha_x^{18}, \beta_y^{18}, \gamma_z^{18}$
Right clavicle orientation (9)	2	α_x^9, γ_z^9	Right knee orientation (19)	1	γ_z^{19}
Right shoulder orientation (10)	3	$\alpha_x^{10}, \beta_y^{10}, \gamma_z^{10}$	TOTAL	32	

Table 2. The 11 steps of the hierarchical optimisation. Joint indices are the same as in Figure 2.

(Step 1) Global body pos.: 3 DOF: r_x, r_y, r_z	**(Step 2)** Torso: 3 DOF: $\alpha_x^1, \beta_y^1, \gamma_z^1$	**(Step 3)** Head: 2 DOF: α_x^2, γ_z^2
(Step 4) Left upper arm: 4DOF: $\alpha_x^5, \gamma_z^5, \alpha_x^6, \gamma_z^6$	**(Step 5)** Right upper arm: 4DOF: $\alpha_x^9, \gamma_z^9, \alpha_x^{10}, \gamma_z^{10}$	**(Step 6)** Left lower arm: 2DOF: β_y^6, γ_z^7
(Step 7) Right lower arm: 2DOF: $\beta_y^{10}, \gamma_z^{11}$	**(Step 8)** Left upper leg: 4DOF: $\alpha_x^{13}, \gamma_z^{13}, \alpha_x^{14}, \gamma_z^{14}$	**(Step 9)** Right upper leg: 4DOF: $\alpha_x^{17}, \gamma_z^{17}, \alpha_x^{18}, \gamma_z^{18}$
(Step 10) Left lower leg: 2DOF: $\beta_y^{14}, \gamma_z^{15}$	**(Step 11)** Right lower leg: 2DOF: $\beta_y^{18}, \gamma_z^{19}$	

4.4 Fitness function

The fitness function compares the silhouettes generated by the model in its candidate pose with the silhouettes extracted from the original images. The original images can be acquired from N different viewpoints. Each image is foreground-background segmented and binarised to obtain a silhouette. Let the images containing the *original* silhouettes be denoted as I_i^o, $i = 1...N$. Similarly, let I_i^m, $i = 1...N$ denote images of the *model* silhouettes. The cost function can then be written as follows:

$$E = \sum_{i=1}^{N} \frac{1}{Z_i} \sum_{1}^{row} \sum_{1}^{col} (I_i^o \ \& \ I_i^m), \tag{5}$$

where row and col denote the image rows and columns, respectively, and $\&$ denotes the bitwise *AND* operation. Coefficients Z_i are the normalisation constants obtained by counting the number of silhouette pixels in every original image.

5 Experiments

Data. The set of 5 test sequences are a courtesy of CSSVP, University of Surrey. They were acquired in a dedicated multi-camera acquisition studio and consist of 10 synchronised videos with resolution 720×576, acquired at 25fps.

Table 3. This table shows the consistency of the joint pose estimates for each of the 5 test sequences over 10 runs of the pose estimation algorithm. As the mean joint position estimate depends on the pose which changes through the sequence, we only report, for each joint, the standard deviation (in cm) in the estimate of its 3-D position computed over the entire sequence. Joint numbers correspond to those shown in Figure 2.

Sequence	Jon Walk			Tony Kick			Tony Punch			Tony Stance			Tony Walk		
Joint Number	σ_x	σ_y	σ_z	σ_x	σ_y	σ_z	σ_x	σ_y	σ_z	σ_x	σ_y	σ_z	σ_x	σ_y	σ_z
1	2.4	1.1	1.9	1.8	1.1	1.6	0.6	0.5	0.6	0.9	0.6	0.9	2.7	0.9	2.3
2	1.1	1.1	1.0	1.6	0.9	1.2	0.5	0.4	0.8	0.7	0.6	0.7	1.5	0.9	1.1
3	0.7	1.0	0.7	0.9	0.9	0.7	0.2	0.4	0.4	0.4	0.6	0.4	0.8	0.9	0.6
4	0.2	1.0	0.4	0.3	0.8	0.4	0.1	0.5	0.2	0.2	0.6	0.3	0.2	0.9	0.2
6	1.2	1.3	3.1	2.0	1.2	3.6	0.7	0.6	2.1	0.8	0.8	3.0	1.7	1.7	3.0
7	1.8	4.1	1.8	1.3	1.0	3.0	0.5	0.6	1.0	0.6	0.9	1.0	0.9	2.3	2.0
8	2.1	5.8	2.7	1.9	3.6	1.8	1.6	4.5	2.6	1.4	5.3	3.8	1.1	2.2	3.0
10	1.2	1.1	3.1	1.8	1.1	3.2	0.9	0.6	1.6	0.9	0.8	2.2	1.6	1.7	1.5
11	0.9	1.5	1.7	1.5	1.4	2.4	0.7	0.5	1.5	0.6	0.4	2.2	0.9	2.3	1.6
12	0.9	1.3	1.6	1.9	1.9	1.4	1.6	1.3	1.1	1.1	1.5	0.6	0.9	2.1	2.3
14	2.4	1.1	1.7	1.8	1.1	1.3	0.6	0.5	0.6	0.9	0.6	1.2	2.7	0.9	2.1
15	0.9	1.3	0.7	0.5	1.1	0.6	0.3	0.5	0.3	0.4	0.6	0.6	1.6	1.3	2.0
16	2.3	1.5	2.5	1.9	1.2	0.8	0.8	0.5	0.3	0.9	0.7	0.5	2.7	1.8	5.3
18	2.4	1.1	1.9	1.8	1.2	1.7	0.6	0.5	0.6	0.9	0.6	0.9	2.7	0.9	2.4
19	0.6	1.3	0.6	1.7	2.3	1.6	0.3	0.5	0.2	0.3	0.7	0.3	1.9	1.2	3.4
20	2.3	1.4	1.4	2.7	3.1	1.7	0.4	0.5	0.3	0.7	0.7	0.5	3.4	2.7	5.3

Algorithm settings. The experiments we report in this paper were run with a swarm containing 10 particles. The PSO inertia parameter decreased over time as in [8, 9], that is, it decreased according to $w = 2.0/e^x$, where x has the role of a counter and where the starting value for the first frame was set to $x = 1.0$ and for all subsequent frames to $x = 2.0$. Whenever a PSO iteration (one swarm move) did not produce an improved global best estimate, the inertia value decreased by increasing the counter to $x = x + 0.05$. The optimisation terminated when the inertia value fell under 0.1. The constants C_1 and C_2 in Equation (2) were set to 2.0.

GPU. The experiments were run with an nVIDIA™ Quadro FX 5800 with 4GB Gddr3 RAM on a PC powered by a 64-bit Intel(R) Core(TM) i7 CPU running at 2.67GHz.

Human Body Model. The process of pose estimation, as presented in this paper, requires that the particle position vector be rendered as a human body model using Catmull-Clark subdivision and then projected onto the camera image plane(s) to generate candidate silhouettes. To perform the body model subdivision on the GPU, we have adapted the implementation by Patney et al. [17]. The projection of the body model onto camera planes is implemented in OpenGL and is the only operation that has been left to the CPU. As such, it represents a bottleneck of our algorithm, because it incurs a memory transfer between the CPU and GPU every time the body models are rendered to generate silhouettes for the fitness function. In order to minimise the number of transfers, we render all camera views for all particles into one large OpenGL buffer and perform only one transfer for every iteration of the PSO. As the OpenGL buffer size is limited to 8192×8192 pix, we can use only 10 particles before exceeding the available buffer size. Porting the camera projection code onto GPU would remove the problem of the CPU-GPU memory transfer and allow the use of larger swarms as the

limit would not be imposed by the OpenGL buffer size, but instead by the amount of memory available on the graphics card.

Results. The presented CUDA-PSO-based pose estimation algorithm was developed from the hierarchical PSO reported in [8] for the problem of upper body pose estimation and using a subdivision surface body model. The same algorithm was later adapted to full body pose in [9]; however, it also replaced the subdivision model with a simpler cylinder model to enable a fair comparison of the search method with a competing particle filtering approach. The work was further extended in [7], where an adaptive full-body hierarchical pose estimation (APSO) was reported which dynamically adjusted the search region size in every frame in an attempt to reduce the computation time. In [7], the APSO algorithm took on average 155 seconds for the Tony Kick and Tony Punch sequence, whereas the algorithm reported here requires only 6.9 seconds per frame for the same sequence. Similarly, in [7], Jon Walk required 176 seconds per frame while our algorithm takes only 7.4 seconds. Not only does the algorithm reported in this paper achieve a 20-fold faster execution time, but it does so with a more complex body model which includes the subdivision process and allows for much more flexibility in modelling the shape of the human body.

Figure 1 shows examples of estimated poses for different camera views and different sequences. We performed a quantitative study of the pose estimation accuracy on a 50-frame long synthetic sequence of a kick, the results of which are reported in Figure 3. The plots show that the mean error with respect to the ground truth over 500 estimates are well below 5 cm for individual joints, and below 7 cm for the full pose which is comparable or better than the competing generative pose estimation methods which have been extensively tested in [9]. The main deviation from the ground truth is detected in the right ankle joint (joint 17) in Figure 3 left which is also the reason for the large spread of estimates in Figure 3 right between frames 20 and 25, when the ankle joint is not correctly estimated. In spite of occasional glitches, the optimisation seems to recover from bad estimates without difficulty. The results were obtained with 10 particles and we anticipate that a larger number of particles would further improve the performance; however, this would require the camera projection implementation on GPU and has been left as future work.

As we do not have the ground truth available for the real sequences, we instead study the variability in the pose estimates over 10 runs of the algorithm. The results are shown in Table 3 and indicate that, just like in the synthetic sequence, the estimates are generally consistent with occasional imperfections which, however, do not cause algorithm divergence.

Unlike the competing approaches, our method handles initialisation automatically. We start from a canonical "T-pose" and use a higher starting inertia value in the first frame of the sequence, which causes the particles to explore a larger region of the search space. In the subsequent frames, the temporal consistency of the human motion is exploited by initialising the search around the final estimate of the previous frame and using a lower starting inertia value to encourage the search around the previous estimate. The performance on the first frame is comparable to the performance on the rest of the sequence and in line with the ability of the algorithm to automatically recover from bad estimates. This ability is due to the global search nature of the PSO approach.

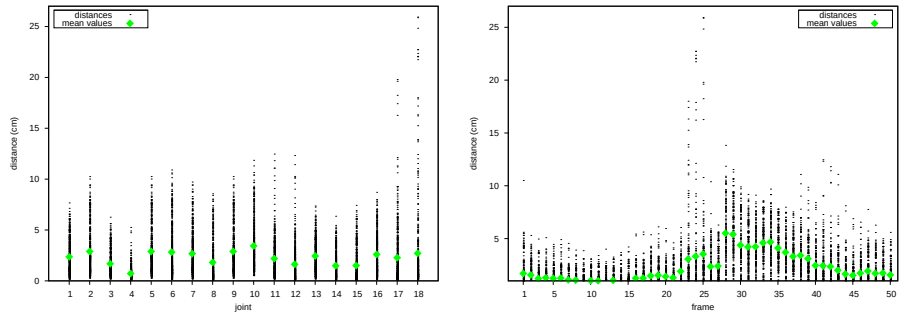

Fig. 3. Algorithm performance on the synthetic sequence. Left: distances (in cm) from the ground truth of each joint estimate in 50 frames over 10 runs. Right: distances from the ground truth of all joint estimates over 10 runs for each of the 50 frames. Means are represented by bullets.

6 Conclusions

In this paper, we described a parallel approach to articulated human body pose estimation from multi-view video sequences, based on the CUDA™ architecture. The results show that the execution time can be cut down noticeably by formulating the algorithm on the GPU, without sacrificing the pose estimation accuracy, thereby exploiting the vast computational resource available on an ordinary desktop PC. The current implementation still combines the computational power of the CPU and GPU and additional speedup is possible by deploying the complete algorithm on GPU in order to avoid the communication bottleneck. This would also allow us to increase the size of the swarm, which is likely to lead to better performance. A further improvement is anticipated from exploiting the parallelism in the kinematic structure of the human body. Both improvements have been left as future work.

Acknowledgments

The authors would like to thank Prof. A. Hilton from the CSSVP, University of Surrey, for the test sequences, and Mr A. Patney from University of California, Davis, for sharing his CUDA implementation of the Catmull-Clark subdivision. S. Ivekovic would like to thank the RSE Lessells Scholarship for the financial support that enabled this work.

References

1. Bandouch, J., Engstler, F., Beetz, M.: Evaluation of hierarchical sampling strategies in 3D human pose estimation. In: Proc. British Machine Vision Conference (2008)
2. Caillette, F., Galata, A., Howard, T.: Real-time 3-D human body tracking using learnt models of behaviour. Computer Vision and Image Understanding 109(2), 112–125 (2008)
3. Chang, J.F., Chu, S.C., Roddick, J.F., Pan, J.S.: A parallel particle swarm optimization algorithm with communication strategies. J. Inf. Sci. Eng. 21(4), 809–818 (2005)
4. Deutscher, J., Reid, I.: Articulated body motion capture by stochastic search. International Journal of Computer Vision 61(2), 185 – 205 (2005)

5. Gall, J., Rosenhan, B., Brox, T., Seidel, H.P.: Optimization and filtering for human motion capture. International Journal of Computer Vision 87(1–2), 75–92 (2010)
6. Gies, D., Rahmat Samii, Y.: Reconfigurable array design using parallel particle swarm optimization. In: Intl. Symp. Antennas and Propagation Soc. vol. 1, pp. 177–180 (2003)
7. Ivekovic, S., John, V., Trucco, E.: Markerless multi-view articulated pose estimation using adaptive hierarchical particle swarm optimisation. In: Proceedings of EvoApplications 2010, LNCS 6024. pp. 241–250 (2010)
8. Ivekovic, S., Trucco, E., Petillot, Y.: Human body pose estimation with particle swarm optimisation. Evolutionary Computation 16(4), 509–528 (2008)
9. John, V., Trucco, E., Ivekovic, S.: Markerless human articulated tracking using hierarchical particle swarm optimisation. Image and Vision Computing In Press, Corrected Proof, available online (2010)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. IEEE Int. conf. on Neural Networks. vol. IV, pp. 1942–1948. IEEE CS Press (1995)
11. Li, J., Wang, X., He, R., Chi, Z.: An efficient fine-grained parallel genetic algorithm based on GPU-accelerated. In: IFIP Int. Conf. on Network and Parallel Computing Workshops. pp. 855–862 (2007)
12. MacCormick, J., Isard, M.: Partitioned sampling, articulated objects, and interface-quality hand tracking. In: Proceedings of ECCV, LNCS 1843. pp. 3–19. Springer-Verlag (2000)
13. Moeslund, T., Hilton, A., Krüger, V.: A survey of advances in vision-based human motion capture and analysis. Computer Vision and Image Understanding 104(2-3), 90–126 (2006)
14. Mussi, L., Cagnoni, S.: Particle swarm for pattern matching in image analysis. In: Artificial life and evolutionary computation, pp. 89–98. World Scientific, Singapore (2010)
15. Mussi, L., Daolio, F., Cagnoni, S.: GPU-based road sign detection using particle swarm optimization. In: IEEE Conf. Intelligent System Design and Applications. pp. 152–157 (2009)
16. Organic Motion: http://www.organicmotion.com/ (2010)
17. Patney, A., Ebeida, M.S., Owens, J.D.: Parallel view-dependent tessellation of Catmull-Clark subdivision surfaces. In: Proc. Conf. on High Performance Graphics. pp. 99–108 (2009)
18. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization: an overview. Swarm Intelligence 1(1), 33–57 (2007)
19. Poppe, R.: Vision-based human motion analysis: An overview. Computer Vision and Image Understanding 108(1–2), 4–18 (2007)
20. Schutte, J.F., Reinbolt, J.A., Fregly, B.J., Haftka, R.T., George, A.D.: Parallel global optimization with the particle swarm algorithm. J. Num. Methods in Eng. 61, 2296–2315 (2003)
21. Urtasun, R., Fleet, D.J., Hertzmann, A., Fua, P.: Priors for people tracking from small training sets. In: Proceedings of IEEE ICCV. pp. 403–410 (2005)
22. Veronese, L.d., Krohling, R.A.: Swarm's flight: accelerating the particles using C-CUDA. In: IEEE Congress on Evolutionary Computation (CEC 2009). pp. 3264–3270 (2009)
23. Vicon Motion Capture Systems: http://www.vicon.com/ (2010)
24. Waintraub, M., Schirru, R., Pereira, C.: Multiprocessor modeling of parallel Particle Swarm Optimization applied to nuclear engineering problems. Progress in Nuclear Energy 51, 680–688 (2009)
25. Wang, W., Hong, Y., Kou, T.: Performance gains in parallel particle swarm optimization via nVIDIA GPU. In: Workshop on Computational Mathematics and Mechanics 2009 (2009)
26. Warren, J., Schaefer, S.: A factored approach to subdivision surfaces. Computer Graphics and Applications 24(3), 74 – 81 (2004)
27. Xue, S.D., Zeng, J.C.: Parallel asynchronous control strategy for target search with swarm robots. International Journal of Bio-Inspired Computation 1(3), 151 – 163 (2009)
28. Zhou, Y., Tan, Y.: GPU-based parallel particle swarm optimization. In: Proc. 2009 IEEE Congress on Evolutionary Computation (CEC 2009). pp. 1493–1500 (2009)