GPU-Accelerated Tracking of the Motion of 3D Articulated Figure

Tomasz Krzeszowski, Bogdan Kwolek, and Konrad Wojciechowski

Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-008 Warszawa
`http://www.pjwstk.edu.pl`

Abstract. This paper presents methods that utilize the advantages of modern graphics card hardware for real-time full body tracking with a 3D body model. By means of the presented methods the tracking of full body can be performed at frame-rates of 5 frames per second using a single low-cost moderately-priced graphics card and images from single camera. For a model with 26 DOF we achieved 15 times speed-up. The pose configuration is given by the position and orientation of the pelvis as well as relative joint angles between the connected limbs. The tracking is done through searching for a model configuration that best corresponds to the observed human silhouette in the input image. The searching is done via particle swarm optimization, where each particle corresponds to some hypothesized set of model parameters.

1 Introduction

The era of increase of performance of single-threaded programs at single silicon chip has come to an end. Programs will only increase the performance if they utilize parallelism. Recently, leading GPU vendors make available GPU programming environments. For instance, NVIDIA introduced CUDA environment to perform data-parallel calculations on GPU. As a result, there have been reported several studies in the literature that exploited GPUs for accelerating algorithms, including image processing and recognition algorithms. GPUs provide the best cost-per-performance parallel architecture for data-level parallelism with high computing demands. The performance bottleneck of most implementations intended for execution on GPU is memory access. Therefore, the algorithms to be executed on GPU should be carefully designed in order to achieve good memory performance, which leads to considerable speed-up of the computations. Thus, GPUs are not the best choice for all computer vision problems.

Non intrusive human body tracking is a key issue in advanced human-computer communication. This is one of the most challenging problems in computer vision being simultaneously one of the most computationally demanding tasks. For example, a tracker [1] employing 10 annealing layers with 200 particles needed around 1 hour to process 5 seconds of footage. Considerable amount of work has been done to achieve reliable and fast articulated motion tracking [2][1][3][4][5]. However, to the best of our knowledge, no GPU implementation of articulated body tracking has been developed until now.

2 Programming of GPU

In this Section we discuss the architectural features of G80, which are most relevant to understand our implementation. The G80 graphics processing unit architecture was first introduced in NVIDIA's GeForce 8800 GTS and GTX graphics cards. A GTX 280 card that is compatible with G80 and supports Computing Capability 1.3 has been used in our experiments. It has 240 cores in 30 streaming 1.3 GHz multiprocessors, which support Single Program Multiple Data (SPMD) programming model.

The programming of GPU has been considerably simplified through introducing CUDA framework by NVIDIA. CUDA makes programming of GPU easier as it hides hardware details allowing a programmer to think in terms of memory and arithmetic operations, rather than in categories of primitives and textures being specific to graphical operations. To obtain the best performance from G80-based GPUs, we have to keep all processors occupied and hide memory latency. In order to achieve this aim, CUDA supports running hundred or thousands of lightweight threads in parallel. No extra code is needed for thread management, and the CPU is capable of running concurrently with the GPU. In CUDA the programs are expressed as kernels. A part of the application that operates on different elements of a dataset can be isolated into a kernel that is executed on the GPU by many different threads. Kernels run on a grid, which is an array of blocks, whereas each block is an array of threads. Blocks are mapped to multiprocessors and each thread is mapped to a single core. Threads within a block are grouped into warps.

At any time a multiprocessor can execute a single warp. Every thread of a warp executes the same instruction but operates 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. Threads in one block can communicate each other using the shared memory, but two threads from two different blocks cannot cooperate via shared memory. The GPU handles latency by supporting thousands of threads in flight at once. In current GPUs, context switch is very fast because everything is stored in registers and thus there is almost no data movement. The card's DRAM memory is accessible from different blocks. It is, however, much slower than the on-chip shared memory. Its latency can be hidden by careful design of control flow as well as design of kernels. To achieve good performance both high density of arithmetic instructions per memory access as well as several hundreds of threads per block are needed. This permits the GPU to execute arithmetic instructions while certain threads are waiting for access to the global memory.

3 Parallel PSO for Object Tracking

Particle swarm optimization (PSO) [6] is a global optimization, population-based evolutionary algorithm for dealing with problems in which a best solution can be represented as a point in a n-dimensional space. The PSO is initialized with a group of random particles (hypothetical solutions) and then it searches hyperspace (i.e. R^n) of a problem for optima. Particles move through the solution space, and undergo evaluation according to some fitness function after each time step. The particles iteratively evaluate their candidate solutions and remember the location of their best location with the smallest objective value so far, making this information available to their neighbors. Particles communicate good positions to each other and adjust their own velocity and then the position based on such good positions. Additionally each particle employs 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

A topology with the global best converges faster as all the particles are attracted simultaneously to the best part of the search space. Neighborhood best allows parallel exploration of the search space and decreases the susceptibility of falling into local minima, however, it slows down the convergence speed. Taking into account the computational overheads the topology with global best is utilized in our approach.

In the ordinary PSO algorithm the update of particle velocity and position is given 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}.

Given the above equations the PSO algorithm can be illustrated in the following manner:

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* and update. At the beginning of each frame, in the initialization stage an initial position $x^{(i)} \leftarrow \mathcal{N}(p_{\text{g}}, \Sigma)$ 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 predefined observation model as follows:

$$f(x^{(i)}) = p(o^{(i)}|x^{(i)}) \tag{3}$$

where $o^{(i)}$ is the observation corresponding to $x^{(i)}$. It is the most time consuming operation on GPU. The calculation of the observation model is discussed in Section 4.2 and the decomposition of this operation 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 PSO 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 PSO 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 every optimization iteration. In contrast to synchronous PSO the asynchronous algorithm updates particle positions and velocities continuously using currently accessible information.

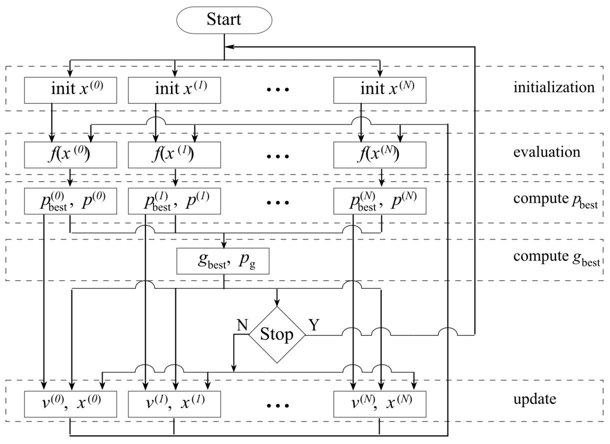
Start
init $x^{(0)}$
init $x^{(1)}$
init $x^{(N)}$
initialization
$f(x^{(0)})$
$f(x^{(1)})$
$f(x^{(N)})$
evaluation
$p_{\text{best}}^{(0)}, p^{(0)}$
$p_{\text{best}}^{(1)}, p^{(1)}$
$p_{\text{best}}^{(N)}, p^{(N)}$
compute p_{best}
$g_{\text{best}}, p_{\text{g}}$
compute g_{best}
N
Stop
Y
$v^{(0)}, x^{(0)}$
$v^{(1)}, x^{(1)}$
$v^{(N)}, x^{(N)}$
update

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

4 Implementation of Articulated Body Tracking on GPU

4.1 Tracking framework

The articulated model of the human body has a form a kinematic chain consisting of 11 segments. The 3d model is constructed using cuboids that model the pelvis, torso, head, upper and lower arm and legs. The configuration of the model is defined by 26 DOF. It is determined by position and orientation of the pelvis in the global coordinate system and the relative angles between connected limbs. Each cuboid can be projected into 2d image plane via perspective projection. To obtain a projected image of the model we transform the corners via perspective projection and afterwards we perform a rendering of the cuboids. This way we obtain the image of the 3d model in a given configuration. During calculation of the fitness function we employ a regular rectangular grid to extract pixel values for each body part in such a rendered image.

In most of the approaches to articulated object tracking a background subtraction algorithms are employed to extract a person undergoing tracking. Additionally, image cues such as edges, ridges, color are often employed to improve the extraction of the person. In this work the images of the person undergoing tracking are synthesized on the basis of 3d models of the human body. The parameters are determined using model in a configuration, which has been manually determined through fitting of the model to the person on the input images. Given such parameters we generate synthesized images of the human body. In the experiments we employ animations generated via Blender application[1] as well as images that are generated on the basis of the same 3d model that is used in tracking, see sub-images in Fig. 2.

4.2 Observation model

The most computationally and time demanding operation is generation of the body image on the basis of the hypothesized body configurations established by particles. Precisely speaking, the most computationally intensive operation is the rasterization of the triangles. The rendering stage creates a two dimensional display of triangles given the transformed vertexes of the 3d model. It involves the calculation of the pixels forming the triangles. GPU designers have incorporated many rasterizatiom algorithms over the years. In all rasterization algorithms the pixel is treated independently from all other pixels. Therefore, the GPU can handle all pixels in parallel.

The so-called painter's algorithm consists in sorting the object or polygons from back to front and then rasterizing them in that order. Currently, a modified painter's algorithm is used to perform the depth test. In the parallel rendering algorithm, which is based on a modified version of the painter's algorithm, we perform painting in the reverse order, i.e. we first paint out the nearest element. Afterwards, we paint out the triangles according to the order of the model parts.

[1] http://www.blender.org/

In order to paint out a given triangle we determine the surrounding rectangular sub-image and then we verify all pixels of such a sub-image. If a considered pixel had not been previously painted out we verify if it belongs to the considered triangle. If yes, we paint it out.

The fitness function (3) is determined on the basis of the overlap degree between the reference image of the body and the current rasterized image. The overlap degree is calculated through checking the overlap from the reference to the current rasterized image as well as from the current rasterized image to the reference body. The larger the degree overlap is, the larger is the fitness value. Figure 2 depicts some images used in the experiments. The images were acquired by surveillance cameras in a student hostel. In the sub-images at the bottom-left the reference images are shown.

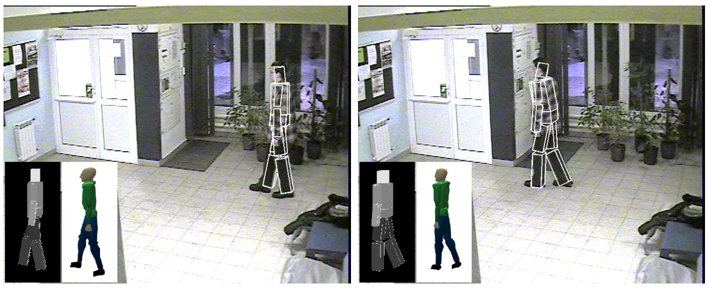

Fig. 2. 3d model-based human body tracking, frames #5, #40, left bottom: appearance images of person undergoing tracking. The overlap degree between the appearance image and the projected model into 2d image plane is 0.84 and 0.86, respectively.

4.3 Algorithm decomposition

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 stage we generate pseudo-random numbers using the Mersenne Twister [7] kernel provided by the CUDA^{TM} SDK. From uniform random numbers we generate a vector of normal random numbers using Box Mueller transform based on trigonometric functions [8] to initialize the positions of the particles. At the beginning of each iteration we generate the random numbers for all particles through single call of the kernel. Taking into account that the maximum number of threads in one block is 512, in one block we initialize 19 particles. In the evaluation phase we employ two kernels. The first kernel is used in rendering of the 3d body model into 2d plane, whereas the second one in calculation of the measure similarities between projections of the 3D model and the content of the reference images. In our approach each block is responsible for rendering one image. Taking into account the available number of registers we run 448 threads and each thread is in charge of painting out of several pixels. In order to obtain the degree of overlap the comparison of the images is done using one dimensional textures and a single thread compares the pixels from two corresponding image columns. In the update phase each thread is responsible for updating one dimension of the particle's location.

5 Experiments

In this Section, we first compare the runtimes of our GPU and CPU implementations and present our speedup. Then, we show the tracking performance using synchronous and asynchronous implementations of PSO. This is followed by a discussion of the factors that limit our performance.

The experiments were conducted on a notebook with 4 GB RAM, Intel Core 2 Duo, 2 GHz processor with GT 130M graphics card. The graphics card has 4 stream multiprocessors with 1.5 GHz, each with 8 cores. It is equipped with 512 MB RAM, 64 KB constant memory and 16 KB common memory. We conducted also experiments on a PC with single NVIDIA GTX 280 card. The card has 30 stream multiprocessors with 1.3 GHz clock, each with 8 cores. It has 1 GB RAM, 64 KB constant memory and 16 KB common memory.

Table 1 shows computation time that has been obtained on CPU, GT 130M and GTX 280. Using the PSO algorithm with 500 particles and 5 iterations we can process in real-time 5 frames per second. The average degree of overlap between the reference body image and the projected body with the estimated configuration in the 50 frame long sequence is slightly below 0.8. The results in table demonstrate that the mobile graphics card was also capable of obtaining a speed-up.

Table 1. Computation time [sec.]

	CPU	GT 130M	GTX 280
#4000, 10 it.	48.89	20.35	2.94
#2000, 10 it.	24.51	10.06	1.49
#1000, 10 it.	12.28	5.26	0.75
#500, 10 it.	6.12	2.65	0.39
#4000, 5 it.	26.74	11.19	1.59
#2000, 5 it.	13.38	5.52	0.81
#1000, 5 it.	6.68	2.87	0.41
#500, 5 it.	3.34	1.45	0.22

We compared the effectiveness of the synchronous and asynchronous version of the PSO algorithm. The asynchronous PSO that is used in our CPU implementation gives something better results. For instance, for a set-up with 2000 particles and 10 iterations the overlap degree for asynchronous PSO is equal to 0.85, whereas for synchronous version it is equal to 0.80. In a set-up with 500 particles and 10 iterations the overlap degree is equal to 0.79 and 0.78, respectively.

The most time-consuming operation of the tracking algorithm is the rendering of the 3d model. This operation amounts to 0.92 of whole processing time. The comparing of images in order to determine the degree of overlap amounts to 0.05 of full amount of processing time.

6 Conclusions

In this paper we presented an algorithm for articulated human motion tracking on GPU. The articulated model of the human body consists of 11 segments and has 26 DOF. We showed that our GPU implementation has achieved a speedup of more fifteen times than our CPU-based implementation. The tracking of full body can be performed at frame-rates of 5 frames per second using a single low-cost graphics card and single camera images. With rapid development of the graphics card technologies, the tracking speed is expected to be further accelerated in the near future by newer generations of the GPU architecture.

Acknowledgment

This paper has been supported by the project “System with a library of modules for advanced analysis and an interactive synthesis of human motion” co-financed by the European Regional Development Fund under the Innovative Economy Operational Programme - Priority Axis 1. Research and development of modern technologies, measure 1.3.1 Development projects.

References

1. Deutscher, J., Blake, A., Reid, I.: Articulated body motion capture by annealed particle filtering. In: IEEE Int. Conf. on Pattern Recognition. (2000) 126–133
2. Poppe, R.: Vision-based human motion analysis: an overview. Computer Vision and Image Understanding **108** (2007) 4–18
3. Fritsch, J., Schmidt, J., Kwolek, B.: Kernel particle filter for real-time 3D body tracking in monocular color images. In: IEEE Int. Conf. on Face and Gesture Rec., Southampton, UK, IEEE Computer Society Press (2006) 567–572
4. Zhao, T., Nevatia, R., Wu, B.: Segmentation and tracking of multiple humans in crowded environments. PAMI **30** (2008) 1198–1211
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. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Trans. Model. Comput. Simul. **8** (1998) 3–30
8. Box, G.E.P., Muller, M.E.: A note on the generation of random normal deviates. The Annals of Mathematical Statistics **29** (1958) 610–611