Mixing Graphics and Compute for Real-Time Multiview Human Body Tracking

Boguslaw Rymut[2] and Bogdan Kwolek[1]

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

[2] Rzeszów University of Technology
Al. Powst. Warszawy 12, 35-959 Rzeszów, Poland
brymut@prz.edu.pl

Abstract. This paper presents an effective algorithm for 3D model-based human motion tracking using a GPU-accelerated particle swarm optimization. The tracking involves configuring the 3D human model in the pose described by each particle and then rasterizing it in each camera view. In order to accelerate the calculation of the fitness function, which is the most computationally demanding operation of the algorithm, the rendering of the 3D model has been realized using CUDA-OpenGL interoperability. Since CUDA and OpenGL both run on GPU and share data through common memory the CUDA-OpenGL interoperability is very fast. We demonstrate that thanks to GPU hardware rendering the time needed for calculation of the objective function is shorter. Owing to more precise rendering of the 3D model as well as better extraction of its edges the human motion tracing is more accurate.

1 Introduction

In the last decade, GPU computing has considerably evolved to deliver teraflops of floating-point compute power. Many real-time applications require a mix of compute and graphics capabilities, in addition to efficiently processing large amounts of data. Such applications include physically-based simulations, computer vision, augmented and virtual reality, motion capture and visualization, etc. In order to maximize performance, the applications must be designed to allow data to be passed efficiently between compute and graphics contexts. Owing to the CUDA-OpenGL interoperability we can avoid the back and forth data transfer between the host and device memories, and carry out all processing required for 3D rendering and display [10]. CUDA is a parallel computing platform and programming model, which interfaces CPU and of the graphics processing unit, whereas the OpenGL is well known programmer's interface to graphics hardware.

Model-based pose estimation algorithms aim at recovering human motion from one or more camera views and a 3D model representing the human body. An articulated human body can be perceived as a kinematic chain consisting of at least eleven parts corresponding to key parts of the human body. The human pose is typically represented by a vector of joint angles. Typically such a 3D human model consists of very simple geometric primitives like cylinders or truncated cones. The 3D articulated model is projected into each camera view. The majority of 3D motion tracking algorithms are based on minimizing an error function, which measures how well the 3D model projections fit the images. The model - image matching for pose estimation is usually formulated as an optimization of an error/likelihood function. The 3D model rasterization in the cameras' views is the most computationally demanding operation. As demonstrated in [5], a considerable speedup of the tracking can be achieved thanks to parallel computations on GPU, and particularly owing to GPU-based rendering of the 3D model.

Although the general purpose computing on GPU are becoming more popular because of its promise of massive parallel computation, achieving a good performance is still not a simple task. In order to achieve desired performance we have to keep all processors occupied and hide the memory latency. To attain such aim, CUDA supports running hundred or thousands of lightweight threads in parallel. The context switch is very fast because everything is stored in the registers and thus there is almost no data movement.

In general, the image processing and analysis algorithms are good candidates for GPU implementation, since the parallelization is naturally provided by per-pixel operations. Many research studies confirmed this by showing GPU acceleration of many image processing algorithms [1,3,8]. A recent study [7] reports a speedup of 30 times for low-level algorithms and up to 10 times for high-level functions, which contain more overhead and many steps that are not easy to parallelize.

Non intrusive human body tracking is a key issue in user-friendly human-computer communication. This is one of the most challenging problems in computer vision being at the same time one of the most computationally demanding tasks. Particle filters are typically employed to achieve articulated motion tracking. Several improvements of ordinary particle filter were done to achieve fast and reliable tracking of articulated motion [2] as well as to obtain the initialization of the tracking [11]. 3D motion tracking can be perceived as dynamic optimization problem. Recently, particle swarm optimization (PSO) [4] has been successfully applied to achieve human motion tracking [5]. The motion tracking is achieved by a sequence of static PSO-based optimizations, followed by re-diversification of the particles to cover the possible poses in the next time step. The calculation of the matching score is the most time consuming operation of the tracking algorithm. In turn, the rasterization of the 3D model in the cameras' views is the most computationally demanding operation in the evaluation of the fitness function. This motivated us to investigate the feasibility of mixing graphics and compute to speed-up the rasterization of the 3D model, i.e. by the use of CUDA-OpenGL interoperability.

2 Overview of CUDA-OpenGL Interoperability

A GPU is a dedicated processor that offloads 3D graphics rendering workload from the CPU. The synthesis of the image is traditionally implemented as a pipeline of specialized stages, which is called the graphics pipeline. The input to the graphics pipeline is a wireframe consisting of a set of primitives, which are defined by a group of one or more vertices. The basic graphics pipeline stages have evolved from fixed function graphics pipeline into powerful programmable co-processing units capable of performing general purpose computing. One such evolution was introduction of programmable processors consisting of:

– Vertex processor, which aim is to transform each input vertex to data required by the next graphics pipeline stages.
– Geometry processor, which processes a mesh at primitive level and produces vertices and attributes to define new primitives.
– Fragment processor, which determines the color of each fragment.

Such processors could be programmed with programs called shader programs. The three different shader types were merged into one unified shader model with a consistent instruction set across all three processor types. Thus, vertex, geometry, and pixel/fragment shaders became threads, running different programs on the programmable cores. These cores on NVidia platform are called CUDA Cores, and a streaming multiprocessor (SM) is composed of many CUDA cores.

In the CUDA programming model the GPU is treated as a coprocessor that executes data-parallel kernel functions. The kernel is typically executed in parallel through a set of parallel threads, which are grouped into parallel thread blocks and in each block the threads are scheduled for execution in groups containing 32 threads called warps. Each thread block consists of multiple concurrently running threads that can cooperate through shared memory and barrier synchronization. Blocks are mapped to streaming multiprocessors and each thread is mapped to a single core. The SM can issue and execute concurrently two warps.

The GPU consists of an array of SM multiprocessors, each of which is capable of supporting thousands co-resident concurrent threads. At each clock cycle, a multiprocessor executes the same instruction on a group of threads within a warp. Programming of SMs is based on SIMD paradigm, where the same instruction is utilized to process different data. In comparison to traditional multicore processors, GPGPUs have distinctly higher degrees of hardware multithreading (hundreds of hardware thread contexts vs. tens), memory architectures that deliver higher peak memory bandwidth (hundreds of gigabytes per second vs. tens), and smaller cache memories.

By using CUDA we can turn a GPU into a powerful image processor, by using OpenGL we can use the same GPU hardware to generate new images. Because CUDA and OpenGL both run on GPU and share data through common memory, the CUDA-OpenGL interoperability is very fast in practice.

3 GPU-accelerated 3D Motion Tracking Using PSO

The motion tracking can by attained by dynamic optimization and incorporating the temporal continuity information into the ordinary PSO [5]. Consequently, it can be achieved by a sequence of static PSO-based optimizations, followed by re-diversification of the particles to cover the potential poses that can arise in the next frame. The re-diversification of the particles can be obtained on the basis of normal distribution concentrated around the best particle location found in the previous frame. The decomposition of the PSO on the available GPU resources has been discussed in [5]. In the following subsection we focus on the evaluation of the cost function since the time needed for the evaluation of the cost function is far larger in comparison time required by PSO-based optimization as well as time needed changing the configuration of the 3D articulated human model. Afterwards, we discuss how to accelerate the computation of the objective function using CUDA-OpenGL interoperability.

3.1 3D Human Body Model

The human body can be represented by a 3D articulated model formed by 11 rigid segments representing the key parts of the body. The 3D model specifies a kinematic chain, where the connections of body parts comprise a parent-child relationship, see Fig. 1. The pelvis is the root node in the kinematic chain and at the same time it is the parent of the upper legs, which are in turn the parents of the lower legs. In consequence, the position of a particular body limb is partially determined by the position of its parent body part and partially by its own pose parameters. In this way, the pose parameters of a body part are described with respect to the local coordinate frame determined by its parent. The 3D geometric model is utilized to simulate the human motion and to recover the persons's position, orientation and joint angles. To account for different body part sizes, limb lengths, and different ranges of motion we employ a set of pre-specified parameters, which express typical postures. For each degree of freedom there are constraints beyond which the movement is not allowed. The model is constructed from truncated square pyramids and is used to generate contours,

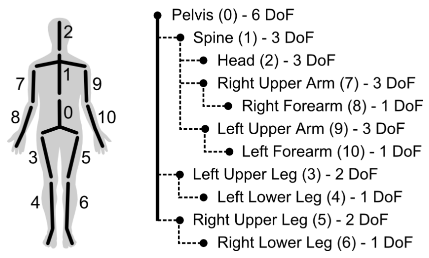
2
1
7
9
8
0
10
3
5
4
6
Pelvis (0) - 6 DoF
Spine (1) - 3 DoF
Head (2) - 3 DoF
Right Upper Arm (7) - 3 DoF
Right Forearm (8) - 1 DoF
Left Upper Arm (9) - 3 DoF
Left Forearm (10) - 1 DoF
Left Upper Leg (3) - 2 DoF
Left Lower Leg (4) - 1 DoF
Right Upper Leg (5) - 2 DoF
Right Lower Leg (6) - 1 DoF

Fig. 1. 3D body model consisting of 11 segments (left), hierarchical structure (right).

which are then matched with the image contours. The configuration of the body is parameterized by the position and the orientation of the pelvis in the global coordinate system and the angles between the connected limbs. The rotation of the limb is done on the basis of the joint transformation matrix. The matrices are calculated on the basis of the joint rotation angles.

3.2 Cost Function

The most computationally demanding operation in 3D model-based human motion tracking is calculation of the objective function. In PSO-based approach each particle represents a hypothesis about possible person pose. In the evaluation of the particle's fitness score the projected model is matched with the current image observations. The fitness score depends on the amount of overlap between the extracted silhouette in the current image and the projected and rasterized 3D model in the hypothesized pose. The amount of overlap is calculated through checking the overlap degree from the silhouette to the rasterized model as well as from the rasterized model to the silhouette. The larger the overlap is, the larger is the fitness value. The objective function reflects also the normalized distance between the model's projected edges and the closest edges in the image. It is calculated on the basis of the edge distance map [9].

The fitness score for i-th camera's view is calculated on the basis of following expression: $f^{(i)}(x) = 1 - ((f_1^{(i)}(x))^{w_1} \cdot (f_2^{(i)}(x))^{w_2})$, where w_1, w_2 denote weighting coefficients that were determined experimentally. The function $f_1^{(i)}(x)$ reflects the degree of overlap between the extracted body and the projected 3D model into 2D image corresponding to camera i. The function $f_2^{(i)}(x)$ reflects the edge distance map-based fitness in the image from the camera i. The objective function for all cameras is determined according to the following expression: $F(x) = \frac{1}{4}\sum_{i=1}^{4} f^{(i)}(x)$. The images acquired from the cameras are processed on CPU. The extracted foreground image and the distance map are then transferred onto the device [9]. Afterwards, they are mapped to the textures and then utilized by the PSO running on the GPU.

3.3 Computing the Cost Function on CUDA-OpenGL

In the phase of the evaluation of the objective function of the PSO running on the GPU we employ two kernels. In the first one, for each individual particle, the 3D model state is converted into global transformation matrix of the hierarchical body model. Each model is projected to the image of each camera, and the total number of the projected models is equal to the number of cameras times the number of the particles. The rasterization of the model is completed by OpenGL hardware on an off-screen frame-buffer. The color components of frame-buffer pixels are mapped to the CUDA texture for the use by the next CUDA kernel. The second kernel calculates the objective function for all particles. In our approach, in every thread block we calculate the fitness score $F(x)$ of single particle. Thus, the number of blocks is equal to the number of the particles, see

Fig. 2. The threads perform the summing of the fitness values of pixels belonging to the image region containing the rasterized model. Taking into account the available number of registers, in each block we run up to 512 threads and each thread is in charge of processing single or several columns of the image depending on the number of the running threads and the image width.

The partial results from each thread of the threaded block are stored in the shared memory and summed using parallel reduction. In each iteration we evaluate and store the sum of two consucitive shared memory cells and thus reduce the number of particpatitng threads by half. The loop is repeated down to complete thread reduction.

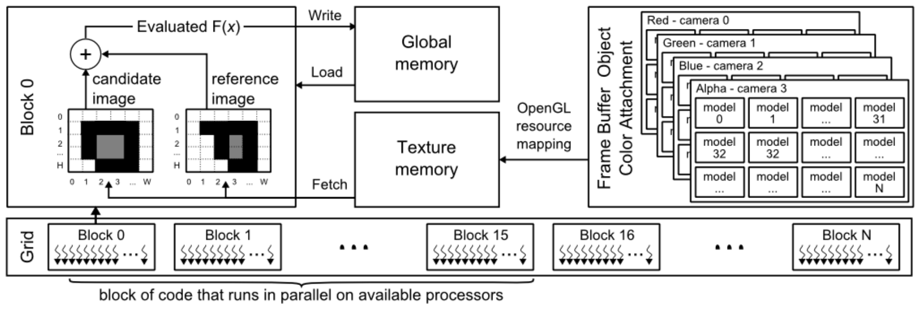

Fig. 2. Evaluation of the cost function using CUDA-OpenGL.

The joint transformation matrices are stored in the Shader Storage Buffer Object (SSBO), whereas the indexes determining the order of painting of the triangles are stored in Index Buffer Object (IBO). The vertices are stored in the Vertex Buffer Object (VBO). The SSBO can be read and written by shaders programs written in GLSL language. The vertex attributes stored in VBO can be accessed by shaders programs. In addition to rendering of the model the programmable processors extract also the edges of the projected models. The OpenGL pipeline is executed twice for each camera, where in the first run the vertex shader projects the vertices from 3D to 2D plane using the Tsai camera model, and the fragment shader draws the visible triangles. In the second run the geometry shader extracts the edges of the projected models, which are then rendered in the frame-buffer. Thus, it stores the model edges in the frame-buffer. All models are rendered simultaneously for a given camera. The rendered images together with the extracted edges of a given camera are stored in one of RGBA components of Frame Buffer Object (FBO). The maximum size of the FBO on the utilized graphics card is 16384×16384 and this in turn allows us to put 32 images in one row of the FBO. The rendered image, which is stored in FBO is mapped to CUDA texture and is employed by a kernel responsible for calculating of the objective function. The SSBO, which is mapped to the linear memory, is used by the first kernel to store the model transformation matrixes, which in turn are used in OpenGL-based rendering.

4 Experimental Results

The experiments were conducted on a PC with Intel 3.46 GHz CPU, 8 GB RAM, and NVidia GeForce GTX 590 graphics card consisting of two CUDA devices. Each CUDA device has 16 multiprocessors and 32 cores per multiprocessor. The card is equipped with 3072 MB VRAM and 48 KB shared memory per multiprocessor. The OpenGL context was created by GLFW and GLEW libraries.

The performance of the algorithm has been evaluated on sequences with walking persons, which were used in our previous work [6]. In particular, in the discussed work above we demonstrated that the average speed-up of GPU over CPU is about 7.5. The images acquired from calibrated and synchronized cameras were preprocessed off-line and transferred frame by frame to the GPU. The input images were rescaled to images of size 480×270. Table 1 shows the average times needed for estimation of the human pose in single frame, obtained by CUDA and CUDA-OpenGL. As we can observe, the time for evaluation of the fitness score using CUDA-OpenGL is far shorter in comparison to time achieved by fitness function implemented in CUDA. It is worth noting that time taken by PSO searching for the best matching is far shorter in comparison to time needed for evaluation of the fitness function and is about 0.9 ms.

Table 1. The average time [ms] of PSO for single frame of size 480×270 (Seq. P1 straight [6]).

	10 it.		20 it.	
# part.	CUDA	CUDA-OpenGL	CUDA	CUDA-OpenGL
64	69.5±3.6	49.4±1.7	135.2±6.9	92.9±3.7
128	67.5±3.5	56.7±2.1	147.6±6.9	119.9±4.9
192	86.6±3.2	70.2±3.0	173.2±7.0	137.6±6.1
256	90.3±3.8	82.6±4.0	179.9±7.0	163.7±8.0

Figure 3 depicts the accuracy of the 3D motion tracking for PSO with various number of particles and iterations, which was obtained on the sequence with a walking person [9]. As we can see, for PSO with the number of particles greater or equal to 128 and the number of iterations equal to 15 or 20 the average error of 3D motion recovery is about 55 mm. The accuracy is slightly better in comparison to accuracy obtained in [9]. In particular, it is better for small number of particles. The results are better due to more precise rendering and different 3D model, since in [9] the model was built on truncated cones.

5 Conclusions

In this work we presented an algorithm for articulated human motion tracking. The tracking has been done in real-time using a parallel PSO that is executed

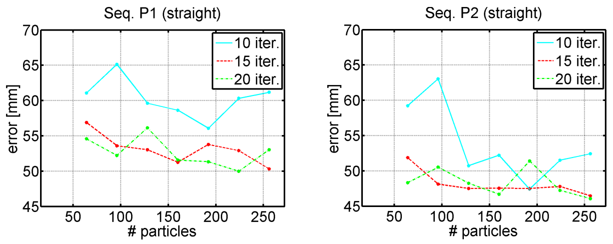

Fig. 3. Tracking errors [mm] versus particles number for CUDA-OpenGL PSO.

on GPU. To accelerate the evaluation of the fitness function, which is the most computationally intensive operation of the tracking algorithm, the rendering of the 3D model has been realized using CUDA-OpenGL. We showed that thanks to rendering of the 3D model using GPU hardware the rendering time is shorter.

Acknowledgment. This work has been partially supported by the Polish Ministry of Science and Higher Education within a grant for young researchers (U-530/DS/M) and the National Science Center within the project N N516 483240.

References

1. Castano-Diez, D., Moser, D., Schoenegger, A., Pruggnaller, S., Frangakis, A.S.: Performance evaluation of image processing algorithms on the GPU. Journal of Structural Biology 164(1), 153 – 160 (2008)
2. Deutscher, J., Blake, A., Reid, I.: Articulated body motion capture by annealed particle filtering. In: IEEE Int. Conf. on Pattern Recognition. pp. 126–133 (2000)
3. Fung, J., Mann, S.: Using graphics devices in reverse: GPU-based image processing and computer vision. In: IEEE Int. Conf. on Multimedia and Expo. pp. 9–12 (2008)
4. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. of IEEE Int. Conf. on Neural Networks. pp. 1942–1948. IEEE Press, Piscataway, NJ (1995)
5. Krzeszowski, T., Kwolek, B., Wojciechowski, K.: GPU-accelerated tracking of the motion of 3D articulated figure. LNCS, vol. 6374, pp. 155–162. Springer (2010)
6. Kwolek, B., Krzeszowski, T., Gagalowicz, A., Wojciechowski, K., Josinski, H.: Real-time multi-view human motion tracking using particle swarm optimization with resampling. In: AMDO'2012, LNCS, vol. 7378, pp. 92–101. Springer (2012)
7. Pulli, K., Baksheev, A., Kornyakov, K., Eruhimov, V.: Real-time computer vision with OpenCV. Comm. ACM 55(6), 61–69 (Jun 2012)
8. Rymut, B., Kwolek, B.: GPU-supported object tracking using adaptive appearance models and Particle Swarm Optimization. In: Computer Vision and Graphics, LNCS, vol. 6375, pp. 227–234. Springer (2010)
9. Rymut, B., Kwolek, B., Krzeszowski, T.: GPU-accelerated human motion tracking using particle filter combined with PSO. In: Advanced Concepts for Intelligent Vision Systems, LNCS, vol. 8192, pp. 426–437. Springer Int. Publ. (2013)
10. Stam, J.: What every CUDA programmer should know about OpenGL. In: GPU Technology Conference (2009)
11. Wu, C., Aghajan, H.: Real-time human pose estimation: A case study in algorithm design for smart camera networks. Proc. of the IEEE 96(10), 1715–1732 (2008)