Hong *et al. Hum. Cent. Comput. Inf. Sci.* (2016) 6:23
DOI 10.1186/s13673-016-0079-9

Human-centric Computing and Information Sciences

RESEARCH

Open Access

Plausible mass-spring system using parallel computing on mobile devices

Min Hong[1], Jae-Hong Jeon[2], Hyo-Sub Yum[3] and Seung-Hyun Lee[4*]

*Correspondence:
slee413@hongik.ac.kr
[4] School of Architectural Engineering, College of Science & Technology, Hongik University, 2639, Sejong-ro, Sejong, South Korea
Full list of author information is available at the end of the article

Abstract

Recently the hardware performance of mobile devices have been extremely increased and advanced mobile devices provide multi-cores and high clock speed. In addition, mobile devices have advantages in mobility and portability compared with PC and Console, so many games and simulation programs have been developed under mobile environments. Physically-based simulation is a one of the key issues for deformable object modeling which is widely used to represent the realistic expression of 3D soft objects with tetrahedrons for game and 3D simulation. However, it requires high computation power to plausibly and realistically represent the physical behaviors and interactions of deformable objects. In this paper, we implemented parallel cloth and mass-spring simulation using graphics processing unit (GPU) with OpenCL and multi-threaded central processing unit (CPU) on a mobile device. We applied CPU and GPU parallel computing technique into spring force computation and integration methods such as Euler, Midpoint, 4th-order Runge-Kutta to optimize the computational burden of dynamic simulation. The integration methods compute the next step of positions and velocities in each node. In this paper, we tested the performance analysis for the spring force calculation and integration method process using CPU only, multi-threaded CPU, and GPU on mobile device respectively. Our experimental results concluded that the calculation using proposed multi-threaded CPU and GPU multi-threaded CPU are much faster than using just the CPU only.

Keywords: Mass-spring system, Deformable object, GPU parallel computing

Background

As the hardware performance of mobile devices have been noticeably increased, not only new applications with utilizing the advanced performance of mobile devices have been developed but also many traditional applications that were already developed based on PC environments have been converted to mobile devices [1]. Mobile devices provide big advantages in mobility and portability compared with general PC or workstations, so that many people can use them to access and to handle information almost anytime and anywhere. The hardware performance of mobile devices has become smaller and fast recently, so many people desire to feel the realistic and immersive virtual experience on mobile devices like PC or Console environments [2, 3].

Representation of a 3D world or physically-based simulation requires lots of physical calculations, but the performance of mobile devices is not high enough to perform the numerical problems. To alleviate these problems, researchers have studied several techniques such

as numerical integration, collision detection, collision response, and multi-core central processing unit (CPU) [4, 5]. However most researches have been deeply focused on PC or Console based environments. Many recent mobile devices have graphics processing unit (GPU) with low clock speed, but they provide lots of arithmetic logic unit (ALUs). Basically GPU works in parallel with ALUs. So when GPU is allocated with some tasks, it separates the tasks to a number of ALUs and performs together at the same time. Therefore, when we apply general-purpose computing on graphics processing unit (GPGPU) for numerically complicated calculations, the computational time can be reduced.

In this paper, we provide a guideline for modeling of 3D objects and selecting of integration methods for physically based simulation on mobile device using parallel computing approach. We proposed and implemented the cloth simulation and 3D mass-spring simulation using multi-thread approach and OpenCL library to compute node position and spring force information with CPU and GPGPU parallel computing. We also implemented a new data structure to calculate the spring forces in parallel computing for deformable object simulation which has uneven spring connections on each node. The proposed method utilizes the shared memory which can reduce the transfer latency of simulation data between CPU and GPU.

Related works

Physically-based simulation

Physically-based simulation provides us to present the physically realistic real world behaviors of 3D deformable objects on computer. In physically-based simulation, 3D objects can be classified as rigid and deformable objects. Unlike rigid objects, deformable objects can naturally and plausibly represent the changes of their shape according to the external forces. Many dynamic simulations have been widely applied to achieve the dynamic behaviors of object on user's purpose such as finite element method (FEM), boundary element method (BEM), finite difference method (FDM), finite volume method (FVM), ray deflectors [6], mass-spring system [7], and chain-mail algorithm [8]. However, the general deformable object modeling approaches can be simply classified into two parts: FEM simulation and mass-spring simulation [9]. Unlike FEM simulation, since mass-spring system requires the fast computational cost instead of accurate estimation of object movement, it has been widely applied for real-time animation or simulation of deformable objects. Therefore, mass-spring system has been dominantly used to represent the practical behavior of deformable objects such as cloth, hair, medical simulation, game, and so on [10, 11].

Integration methods for physically-based simulation

The proper estimation of next status of node positions and velocities is an essential functionality for physically-based simulation. This problem can be solved by various numerical integration methods such as explicit Euler, implicit Euler, semi-Euler, verlet, midpoint, and n-order Runge-Kutta integration method [10]. To use these methods, we have to compute an acceleration α_i that is calculated using Newton's second law of motion. Then using an Eq. (1), we can estimate a position x_i and velocity v_i.

$$a_i(t+\Delta t) = \frac{f_i(t+\Delta t)}{m_i},\ \dot{v}_i = \frac{f_i}{m_i},\ \dot{x}_i = v \tag{1}$$

Here, m_i is mass, x_i is position, v_i is velocity, f_i is force, a_i is acceleration for node. In this paper, we implemented three general integration methods such as semi-Euler, midpoint, and 4th-order Runge-Kutta method. In semi-Euler integration method, position and velocity are updated using Eq. (2).

$$x_i(t + \Delta t) = x_i(t) + \dot{x}_i(t + \Delta t)\Delta t \tag{2}$$

Midpoint integration method improves the basic Euler method by adding a midpoint step and it can improve the calculation accuracy by Eq. (3).

$$x_i(t + \Delta t) = x_i(t) + \dot{x}_i(t) \times \Delta t + \frac{\Delta t}{x}\ddot{x}_i(x) \tag{3}$$

4th-order Runge-Kutta method requires calculating the additional 4 trial steps for approximations of slope by Eq. (4) to guarantee the higher accuracy than midpoint and Euler integration method. Thus 4th-order Runge-Kutta method can provide more accurate solution for the next status, it requires more computational cost for calculation.

$$\begin{aligned} k_1 &= f(x_i(t), t) \times \Delta t,\ k_2 = f\left(x_i(t) + \frac{k_1}{2}, t + \frac{\Delta t}{2}\right) \times \Delta t \\ k_3 &= f\left(x_i(t) + \frac{k_2}{2},\ t + \frac{\Delta t}{2}\right) \times \Delta t,\ k_4 = f(x_i(t) + k_3, t + \Delta t) \times \Delta t \\ x_i(t + \Delta t) &= x_i(t) + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \end{aligned} \tag{4}$$

OpenCL

Open computing language (OpenCL) is a parallel computing framework that is designed for cross platform devices [12]. OpenCL provides the parallel computing using a number of computing devices which can be some processing units (CPUs, GPUs, DSPs, FPGAs and etc.). The parallel computing tasks can be divided into several work groups. Each group consists of many work tasks that are in the basic processing units and they should be executed in a kernel as parallel manner. OpenCL defines a hierarchical memory model as a global memory and local memory. Also OpenCL supports shared memory that is located in the middle of the CPU and GPU and it can efficiently reduce the transfer latency of data between the CPU and GPU [13].

Multi-thread in mobile device

In the multi-thread approach, a number of threads in a process are working simultaneously. Thread is a small action unit which is generated in a process. Thread in mobile device is not directly operated in a running application, but it is used for background process to perform data loading, network communication, and file I/O in the application. The most recent mobile devices contain multiple CPU cores and their calculation speed is also being rapidly upgraded. However, most main logic parts of mobile applications don't efficiently utilize these multiple CPU cores and they have been implemented based on performing in one flow of CPU.

These applications calculate lots of data sequentially, so we can effectively improve the performance of computation when multi-thread approach is applied to the computation process. However, we should consider the data racing to apply multi-threaded CPU

approach. General multi-thread approaches work in the same memory space. Therefore it can cause the wrong data modulation, when several threads are concurrently accessing to the same data. Figure 1 shows an example of data racing problem. Thread 1 and 2 are not synchronized, so we do not know which thread accessed and modulated x value firstly. Therefore, we cannot figure out that what is the final result of x among #1, #2, and #3 task.

To solve this problem, the OS of mobile device suggests several methods to apply multi-thread approach effectively such as pthread for Android and NSThread for iOS, respectively. Therefore, we can apply these multi-thread methods to implement dynamic simulations with CPU multi-core approach [14, 15].

Mass-spring system for parallel computing on mobile device

Cloth simulation

For parallel computing on mobile devices, we can apply the CPU based parallel computing with multi-thread approach and the CPU based parallel computing with OpenCL. The proposed cloth simulation using CPU parallel computing calculates spring forces and node positions with multi-thread computing. Springs in the cloth simulation is divided into n groups using pthread in android and n threads are generated. Each thread performs in parallel and the calculated spring forces are accumulated to their affecting node data. After the parallel computing of spring forces, calculation of node positions using the stored force data is also performed in n parallel groups. Then the calculated node positions of cloth simulation are rendered in GPU and the simulated cloth is displayed on the screen of mobile device. The proposed cloth simulation using GPU parallel

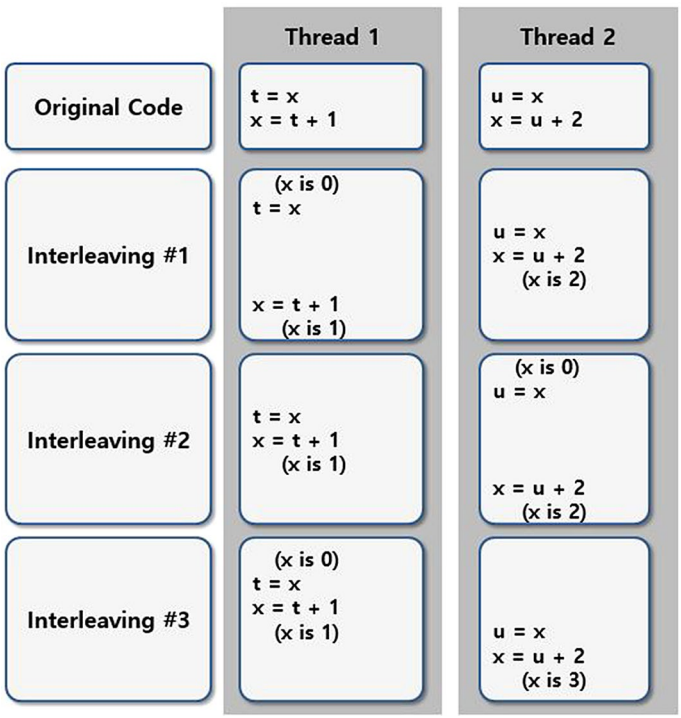

Fig. 1 Example of data racing problem for multi-threaded CPU approach

computing consists of two parts: CPU and GPU computing. The CPU computing part initializes OpenCL and basic cloth simulation properties such as node positions, spring connections, external/internal forces, spring and damping constants. After initializing the CPU part, it is ready to transfer simulation data into GPU using shared memory. When the simulation data is transferred to GPU, the GPU computes all spring forces and sums up into node information. Calculation of spring forces in cloth simulation is somewhat complicated because each node is affected from many connected structural, shear, and bending springs. Figure 2 shows all connection of nodes and springs affecting the cloth simulation.

The proposed cloth simulation with GPU parallel computing should transfer the node information from the CPU to GPU and should be arranged with proper format for GPU. All node information is computed independently on GPU, so GPU has to share the current information of position, velocity, and force. Thus the CPU should provide cloth information array and it should pass this array to GPU using shared memory. When the simulation data is stored in the CPU, OpenCL starts the computation of spring forces and new positions are predicted using a lot of GPU ALUs. Figure 3 shows the flowchart of the proposed parallel cloth simulation using GPU parallel computing. After the calculation of next status of node position and velocity is finished, the proposed cloth simulation process is advanced to the rendering phase directly and then the cloth model is rendered using OpenGL on mobile device [16].

Mass-spring simulation

Although cloth simulation has a standardized number and structure of nodes and springs, most 3D deformable object simulations consist of various numbers of nodes and the springs which are not tightly standardized. Among these object modeling methods, we utilized the tetrahedrons to generate the deformable objects which can express its strain effectively, but it requires a lot of calculations. We implemented mass-spring simulation with the proposed CPU and GPU parallel computing with multi-thread and GPGPU. Figure 4 shows the generated deformable objects with tetrahedrons in our mass-spring simulation and TetGen [17] to generate 3D deformable objects.

The proposed multi-threaded CPU divides whole nodes and springs in an object into n groups and performs the necessary computation separately in each group. In mass-spring simulation, one node in the object can be linked with several other springs, and

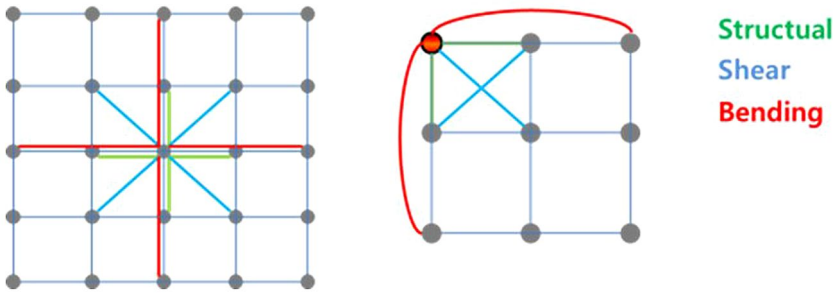

Fig. 2 Structure of node and spring connections

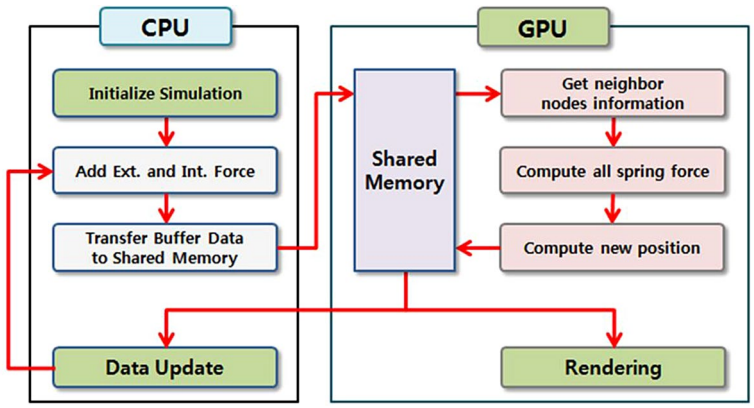

Fig. 3 The proposed cloth simulation using GPU parallel computing

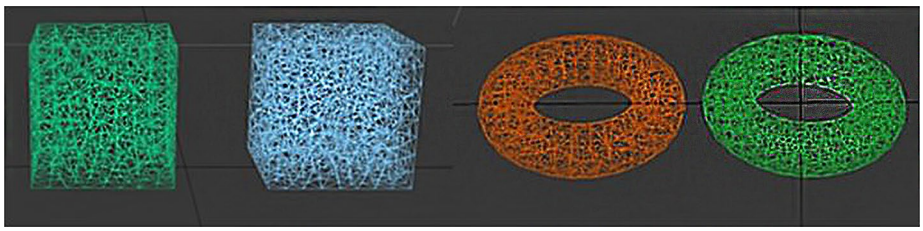

Fig. 4 Deformable objects with tetrahedral mesh

it can be stored improperly due to data racing when the spring forces are calculated in a parallel manner. To prevent this problem, the proposed method utilizes mutual exclusion (mutex) and can apply the critical section. In the critical section, only one thread can access to values and the others should wait until the preoccupied thread is finished. We used lock() and unlock() functions of pthread in android to make the critical section [18]. After spring forces calculation, the proposed method applies additional internal and external force to update the node positions. The node position updating is also calculated with parallel computing. Figure 5 shows the flow of cloth simulation applied multi-threaded CPU with 10 parallel threads.

To utilize GPGPU with OpenCL, it requires modifying the data structure of mass-spring system. Unlike cloth simulation, deformable objects in mass-spring simulation are connected with other nodes un-uniformly by springs, and it is not possible to find the information of other linked nodes. Therefore, we introduce the new data structure to solve this problem. Node position, velocity, rest length, a spring constant, and a damping constant are required to calculate the spring forces in GPU [19]. Thus the CPU has to send node positions and velocity to the shared memory with OpenCL. For spring force calculation, two connected node numbers and the rest length of spring should be grouped as one unit and be sent to GPU. GPU takes this unit and calculates spring force using position and velocity which is delivered by node indexes [20]. Figure 6 shows the proposed data structure of GPU parallel computing for spring force calculation.

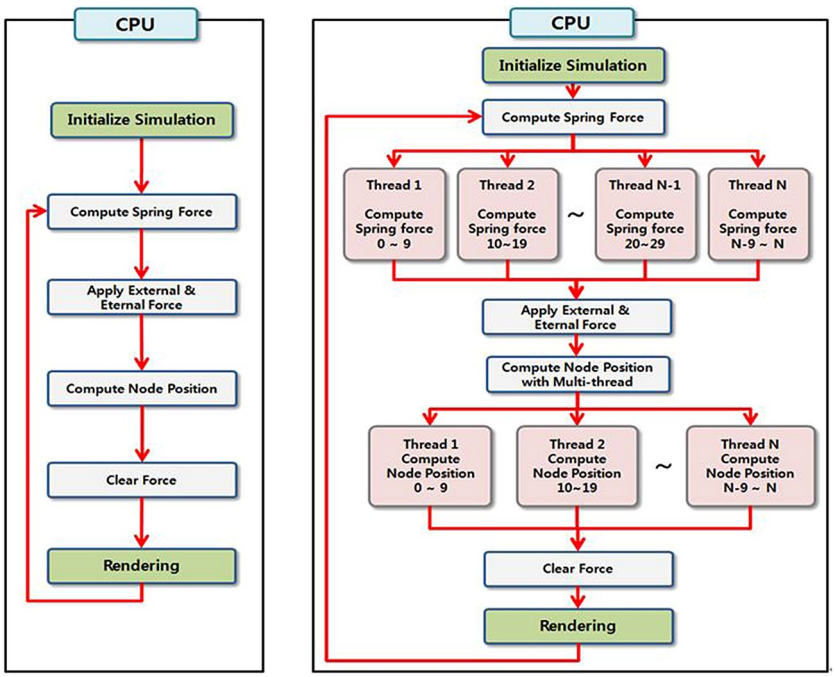

Fig. 5 The proposed cloth simulation using CPU multi-thread

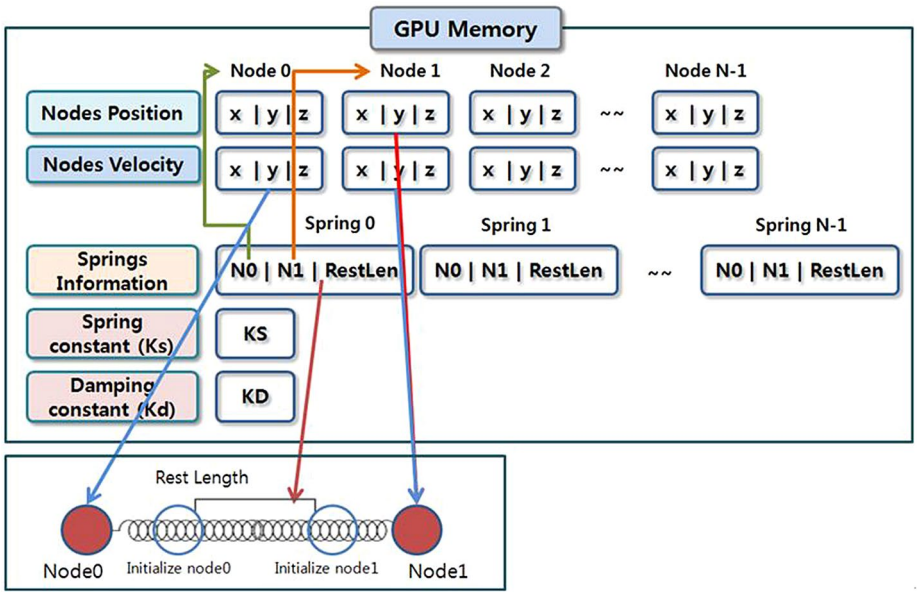

Fig. 6 The proposed data structure for spring force calculation using GPU parallel computing

After force calculation, next status of node position is calculated using the integration method. Since the number of nodes is much less than that of springs in deformable object, GPU parallel computing takes much more time for calculation due to the transfer latency of simulation data between the CPU and GPU. Therefore, the proposed GPU

parallel computing utilizes multi-threaded CPU to calculate the node position. This paper proposed that node positions are calculated using multi-threaded CPU, making it possible to perform parallelism in CPU. Figure 7 depicts the flow of the proposed mass-spring simulation with GPU and multi-threaded CPU.

Simulation results of experimental test

The experimental test for the proposed cloth simulation was performed on the Nexus 7 device. The proposed method was implemented with CPU only, multi-threaded CPU and GPU parallel computing using OpenCL. Since Euler method is sensitive to the time step and can be readily blow-up in the dynamic simulation, we compared the performance of proposed method using 3 types of integrations: semi-Euler, midpoint, and 4th-order Runge-Kutta method. The proposed parallel computing method and the traditional CPU based method are compared with the different number of nodes from 2500 to 350,000 and springs from 29,004 to 4,188,004. In our experiment, some specific nodes in the cloth model were hanged on the virtual ceiling and others are freely falling down towards the floor by gravity which is shown in Fig. 8.

Table 1 shows the performance results of experimental test on millisecond (ms) for every integration steps with CPU only, multi-threaded CPU and GPU with a different level of resolutions. As a result, the order of fast computation cost for integration method was GPU parallel computing, multi-threaded CPU parallel computing, and CPU only. Especially, 4th-order Runge-Kutta method which requires more complex computation to estimate the next status of cloth achieved more advantages with the proposed GPU parallel computing. The proposed GPU parallel computing significantly improved the performance of the cloth.

We also implemented mass-spring simulation for various deformable objects and tested with multi-thread CPU, GPU combined with multi-threaded CPU, and CPU

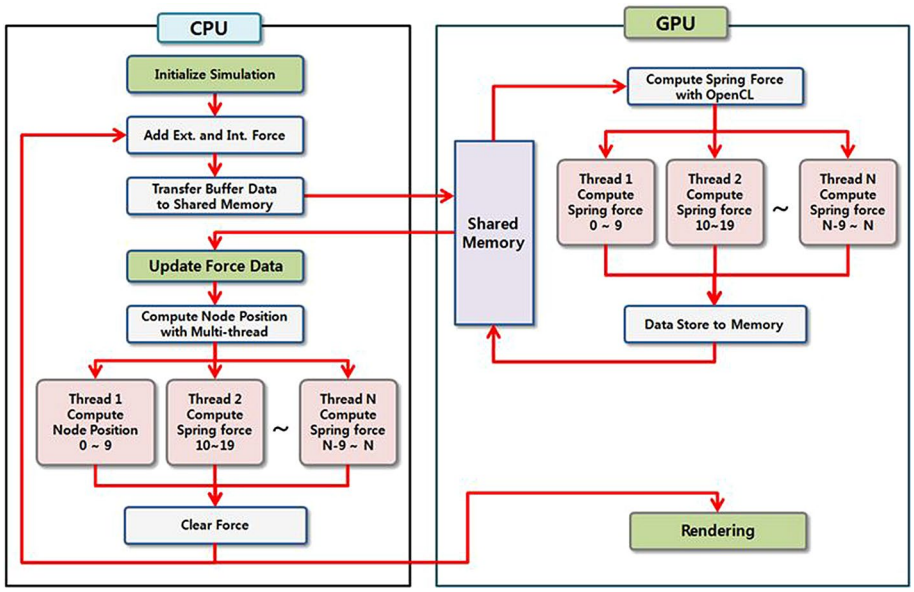

Fig. 7 The proposed mass-spring simulation using GPU and multi-threaded CPU

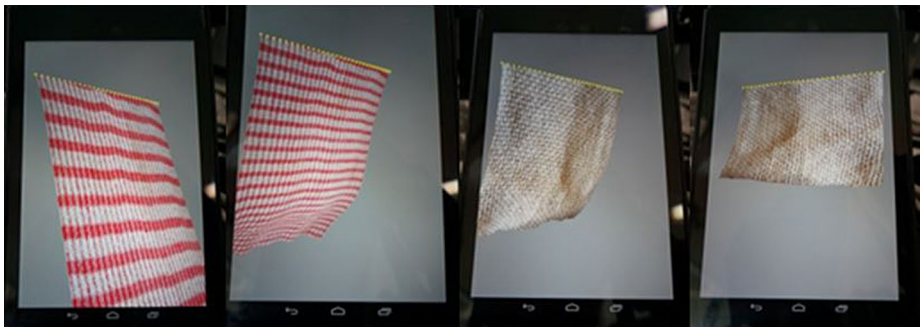

Fig. 8 The snapshots of experimental result for cloth simulation on Nexus 7

Table 1 The experimental performance results for cloth simulation

# of nodes	2500	4900	10,000	40,000	200,000	350,000
# of springs	29,004	57,404	118,004	476,004	2,391,004	4,188,004
Semi-Euler integration (ms)						
CPU only	19.475	41.890	80.480	331.933	1642.693	2733.484
Multi-thread CPU	6.60	13.12	21.98	76.10	349.11	586.35
GPU	1.827	1.963	2.791	20.264	67.130	115.015
Midpoint integration (ms)						
CPU only	20.377	44.047	84.103	345.238	1719.711	2852.775
Multi-thread CPU	7.03	13.50	23.59	79.02	368.26	629.23
GPU	1.735	1.899	2.765	21.986	68.454	117.237
4th-order Runge-Kutta integration (ms)						
CPU only	23.954	50.95	97.763	399.049	1922.532	3335.518
Multi-thread CPU	7.94	15.39	26.70	92.71	453.14	778.48
GPU	1.688	1.793	2.800	22.393	68.117	115.649

only. Mass-spring simulation with proposed method showed improved performance. Deformable objects which are applied for mass-spring simulation are listed in Table 2. As shown in Fig. 9, deformable objects are freely falling down towards the floor by gravity and objects are bounced off the floor. Table 3 shows the result of experimental test on millisecond for each integration step with only CPU, multi-thread CPU and GPU with multi-threaded CPU. As a result, the proposed GPU with multi-threaded CPU parallel computing method was 3–4 times faster than CPU only. Multi-threaded CPU also showed speed 2 times faster than CPU only.

Table 2 Structural information of deformable objects

Objects	Small/medium/large Box	Small/medium/large Sphere	Small/medium/large Torus
# of nodes	700/1579/2941	726/1599/3002	620/962/1977
# of springs	12,921/30,015/58,410	14,145/32,511/60,651	11,070/17,541/39,033

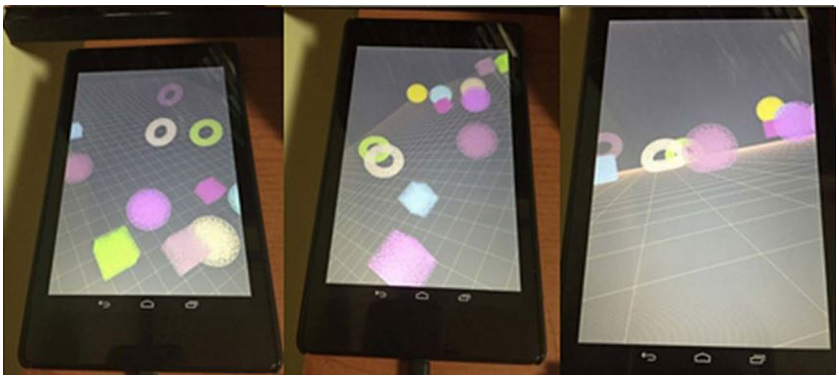

Fig. 9 The snapshots of experimental result for mass-spring simulation on Nexus 7

Table 3 The experimental performance results for mass-spring simulation

# of nodes	3858	4737	7282	10,223	20,587	38,233
# of springs	72,951	90,045	142,662	201,072	408,870	759,330
Semi-Euler integration (ms)						
CPU only	33.43	52.68	83.49	112.92	254.83	469.93
Multi-threaded CPU	24.94	29.74	41.28	95.82	184.28	331.97
GPU + multi-threaded CPU	10.93	15.29	23.22	31.89	68.38	127.45
Midpoint integration (ms)						
CPU only	41.39	58.15	86.82	130.52	245.70	472.29
Multi-threaded CPU	28.84	33.92	53.82	98.53	152.94	282.93
GPU + multi-threaded CPU	11.21	16.84	24.53	34.18	71.82	130.84
4th-order Runge-Kutta integration (ms)						
CPU only	49.24	66.25	94.04	145.29	259.39	513.86
Multi-threaded CPU	30.84	38.13	59.83	103.21	167.32	325.41
GPU + multi-threaded CPU	13.34	19.42	27.92	35.33	73.41	138.98

Conclusions

In this paper, we proposed the multi-threaded CPU and GPU based parallel computing approach with OpenCL library to provide the fast and plausible 3D deformable object simulation on mobile device. To calculate the next status of spring forces, positions, and velocities, we applied and judged three types of numerical integration with semi-Euler, midpoint, and 4th-order Runge-Kutta method based on CPU only, multi-threaded CPU, GPU, and GPU with multi-threaded CPU. The proposed cloth simulation using GPU parallel computing provided around 10–34 times faster than CPU only and utilized the shared memory to communicate the cloth information between CPU and GPU. In addition, the proposed mass-spring simulation using multi-threaded CPU and GPU with multi-threaded CPU provided around 3–4 times faster than CPU only. In this paper, our experimental test provides the guideline for 3D object modeling with mass-spring system to achieve the simulation result in real-time on Nexus 7 device. The proposed mass-spring simulation simultaneously calculates the status of simulation information for one object, thus the calculation of multiple objects is not sufficiently efficient. For the further research, we will study about parallel computing approach to calculate all simulation information of objects at the same time.

Authors' contributions
Author MH and SHL are responsible for the concept of the paper and writing, JHJ and HSY are responsible for the quantitative analysis of the presented results. All authors read and approved the final manuscript.

Author details
[1] Department of Computer Software Engineering, Soonchunhyang University, 22, Soonchunhyang-ro, Asan, South Korea. [2] EZIOT, Suwon, South Korea. [3] Strategy & Planning Team, Bluecore Corporation, 4F, 326 Bongeunsa-ro, Gangnam-gu, Seoul, South Korea. [4] School of Architectural Engineering, College of Science & Technology, Hongik University, 2639, Sejong-ro, Sejong, South Korea.

Acknowledgements
This work was supported by the Soonchunhyang University Research Fund. This paper is an extended version of a conference paper (UCAWSN 2015).

Competing interests
The authors declare that they have no competing interests.

Received: 30 July 2015 Accepted: 28 September 2016
Published online: 28 November 2016

References

1. Cho H, Choi M (2014) Personal mobile album/diary application development. J Converg 5(1):32–37
2. Christou G (2013) A comparison between experienced and inexperienced video game players' perceptions. Hum Cent Comput Inform Sci 3(1):1–15
3. Ho YS (2013) Challenging technical issues of 3D video processing. J Converg 4(1):1–6
4. Saravanan V, Kaushik S, Krishna PS, Kothari DP (2013) Performance analysis of multi-threaded multi-core CPUs. In: The first international workshop. Many-core embedded systems, ACM 2013, Portland, pp 49–53
5. Jeon JH, Hong M, Oh DI, Choi MH (2013) Implementation of 3D deformable objects on smart devices using FFD-AABB algorithm. In: Ubiquitous information technologies and applications. Springer, Berlin, pp 833–840
6. Kurzion Y, Yagel R (1995) Space deformation using ray deflectors. In: Rendering techniques. Springer, Vienna, pp 21–30
7. Gibson SF (1997) 3D Chainmail: a fast algorithm for deforming volumetric objects. In: 1997 Symposium on interactive 3D graphics. pp 149–154
8. Vassilev T, Spanlang B (2002) A mass-spring model for real time deformable solids. In: The east-west vision. pp 149–154
9. Kim JH, Lee YJ (2014) Trivariate B-spline approximation of spherical solid objects. J Inform Process Syst 10(1):23–35
10. Baraff D, Witkin A (1998) Large steps in cloth simulation. In: 25th Conference on computer graphics and interactive techniques, SIGGRAPH 1998. pp 43–54
11. Chen Y, Zhu QH, Kaufman A, Muraki S (1998) Physically-based animation of volumetric objects. In: Computer animation, 1998. pp 154–160
12. The opencl specification https://www.khronos.org/opencl. Accessed Mar 2015
13. Lee H (2015) OpenCL parallel computing for CPUs and GPUs, advanced AMD. http://developer.amd.com/wordpress/media/2013/01/AMD_OpenCL_Tutorial_SAAHPC2010. Accessed Mar 2015
14. Wu CC, Huang JJ (2013) The study of android parallel programming based on the dual-core cortex-A9. In: 2013 ninth International Conference intelligent information hiding and multimedia signal processing. pp 477–480
15. iOS threading programming guide. https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/Multithreading/. Accessed Mar 2015
16. Navalyal GU, Gavas RD (2014) A dynamic attention assessment and enhancement tool using computer graphics. Hum Cent Comput Inform Sci 4(1):1–7
17. Si H (2015) TetGen, a delaunay-based quality tetrahedral mesh generator. ACM Trans Math Soft 41(2):1–36
18. Kosarevsky S, Latypov V (2013) Android NDK game development cookbook. Packt Publishing Ltd, Andheri, p 320
19. Institut National de Recherche en Informatique et Automatique, Provot X (1995) Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In: Proceedings of graphics interface. Canadian Information Processing Society, Québec, pp 147–154
20. Hwang RM, Kim SK, An S, Park DW (2013) The architectural pattern of a highly extensible system for the asynchronous processing of a large amount of data. JIPS 9(4):567–574