# Simulation of Deformable Objects using GLSL 4.3

**Nak-Jun Sung[1], Min Hong[2], Seung-Hyun Lee[3] and Yoo-Joo Choi[4]**
[1] Dept of Computer Science, Soonchunhyang University
Asan, 31538 - Republic of Korea
[e-mail: njsung@sch.ac.kr]
[2] Dept of Computer Software Engineering, Soonchunhyang University
Asan, 31538 - Republic of Korea
[e-mail: mhong@sch.ac.kr]
[3] School of Architectural Engineering, College of Science & Technology, Hongik University
Sejong, 30016 - Republic of Korea
[e-mail: slee43@hongik.ac.kr]
[4] Department of Newmedia Content, Seoul Media Institute of Technology
Seoul, 03925 - Republic of Korea
[e-mail: yjchoi@smit.ac.kr]
*Corresponding author: Yoo-Joo Choi

*Received June 15, 2017; revised July 30, 2017; accepted August 17, 2017;
Published August 31, 2017*

## Abstract

In this research, we implement a deformable object simulation system using OpenGL's shader language, GLSL4.3. Deformable object simulation is implemented by using volumetric mass-spring system suitable for real-time simulation among the methods of deformable object simulation. The compute shader in GLSL 4.3 which helps to access the GPU resources, is used to parallelize the operations of existing deformable object simulation systems. The proposed system is implemented using a compute shader for parallel processing and it includes a bounding box-based collision detection solution. In general, the collision detection is one of severe computing bottlenecks in simulation of multiple deformable objects. In order to validate an efficiency of the system, we performed the experiments using the 3D volumetric objects. We compared the performance of multiple deformable object simulations between CPU and GPU to analyze the effectiveness of parallel processing using GLSL. Moreover, we measured the computation time of bounding box-based collision detection to show that collision detection can be processed in real-time. The experiments using 3D volumetric models with 10K faces showed the GPU-based parallel simulation improves performance by 98% over the CPU-based simulation, and the overall steps including collision detection and rendering could be processed in real-time frame rate of 218.11 FPS.

***Keywords:*** deformable object simulation, GPGPU, parallel graphic processing, GLSL

A preliminary version of this paper was presented at ICONI 2016, and was selected as an outstanding paper.
This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2015R1D1A1A01059304) and was supported by the Soonchunhyang University Research Fund.

## 1. Introduction

Recently, deformable objects that characterized by changing their shape through interaction have been widely used in various fields such as game, animation, and medical simulation. Especially, deformable object's importance is increasing due to development of augmented reality and virtual reality. There are two representative methods for simulating the deformable objects. The first method is mass-spring system, which is mainly applied to real-time simulation. It is difficult to apply the mass-spring system to fields requiring accurate deformation such as medical or architectural fields, while it has been widely applied to various types of interactive games. The second method is the finite element method which is used mainly in the fields to require precise deformation even though it takes a long time because the expression of the deformation is accurate. In this research, we plan to simulate deformable objects using volumetric mass-spring system which can be applied to various applications based on fast performance [1-4]. However, even though the mass-spring system is simple so that it is suitable for real time simulation, it also can require lots of computation time as the complexity of 3D objects increases. Therefore, it is difficult to simulate the deformation of a large model in real time using the traditional CPU single core method. To solve this problem, we need a way to speed up the computation by parallel processing using the GPU.

GPU (Graphic Processing Unit) is a processor that focuses on fast operation through aggregation of ALU (Arithmetic Logic Units) unlike CPU. In the past, the GPU was used only for rendering graphic objects on the screen and user defined programs could not be executed on GPU. However, since the introduction of advanced GPUs such as NVIDIA's GeForce, it has been allowed to execute user-defined geometric operations on GPU. A CPU has a structure in which a small number of ALUs handle a large amount of operations. Instead, a large number of ALUs on GPU can deal with a large amount of computation, which is much faster than the computation through the CPU. In recent, various GPGPU (General-purpose computing on Graphic Process Unit) techniques have been introduced, which enable the computation of general-purpose operations to be speed up using GPU in various fields such as computer vision [5-7]. A user-defined program including graphical operations or general-purpose operations running on the GPU is called a shader. Several shader languages such as Cg, CUDA and OpenCL, have been released. Most of them can be available on specific GPU. To allow shaders to be executed on various kind of GPUs, GLSL (OpenGL Shader Language) 4.3 was released in 2012.

In this paper, we implement shaders for parallel simulation of volumetric deformable objects using GLSL 4.3. In order to perform parallel simulation on GPU, it is important to define data structures suitable for parallel processing. First of all, we construct the deformable objects of regular tetrahedral structures by applying the delauney triangle algorithm inside the existing surface 3D model. Then, the physics rules based on the mass-spring system are applied to the 3D model composed of the divided tetrahedrons for representing natural deformation of objects. For the processing based on the GPU, we need an algorithm that optimized for parallel processing. We also need an algorithm that merges computation results after parallel computation is completed. In this paper, we implement a computational dividing algorithm and a computational result merging algorithm for simulation of multiple deformable objects using compute shader, which is one kind of shaders to be provided in GLSL 4.3.

In Chapter 2 of this paper, research trends on GPU parallel processing and physics based deformation object simulation are investigated and in Chapter 3, the proposed algorithm and data structures for parallel processing are described. Section 4 contains the implementation results and experimental results. Finally, section 5 describes the conclusions and future works related to them.

## 2. Related Work

### 2.1. GPU-based Parallel Processing using GLSL4.3

OpenGL is a cross-platform Open Graphic Library that includes rendering-related functionality. DirectX is available only in Windows environments. However OpenGL can run on a variety of operating systems, such as Linux, MaxOS, and Android. The OpenGL Shading Language (GLSL) is a high-level language that allows access to the GPU pipeline. GLSL is influenced by the versatility of OpenGL so that it can work on various kind of graphics cards. Basically, GLSL supports vertex shader, geometry shader, and fragment shaderw related to graphic rendering. In 2012, a new shader called compute shader has been added to OpenGL 4.3 [8].

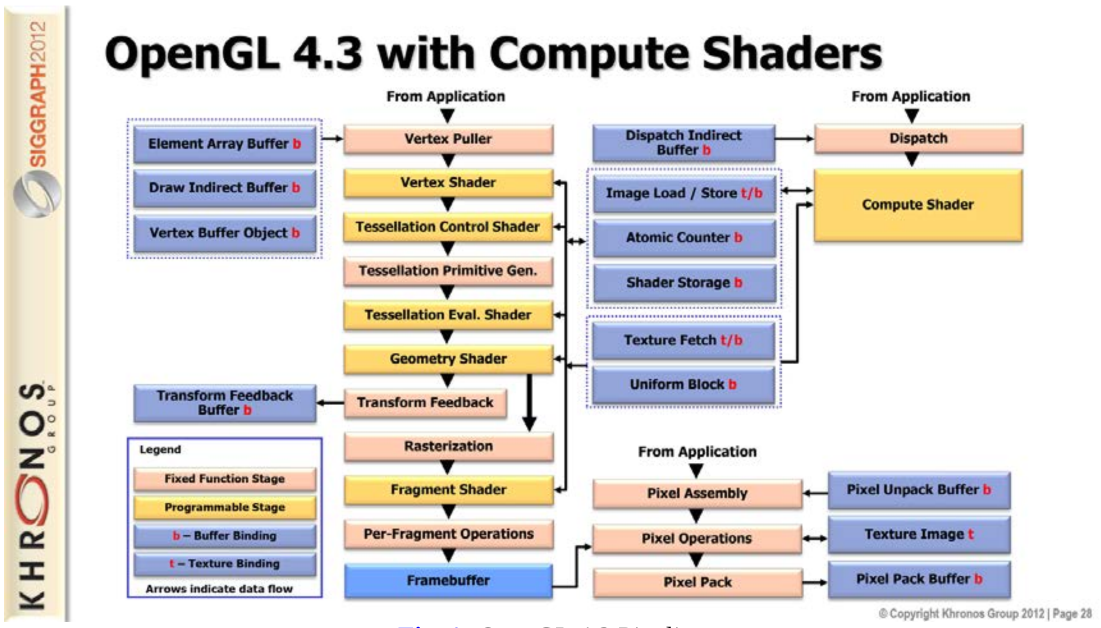

**Fig. 1.** OpenGL 4.3 Pipeline

The compute shader runs on a separate pipeline as shown in **Fig. 1** and it is used to calculate arbitrary information that is not related to the graphics rendering. The compute shader can allocate workloads to several work groups in order to perform parallel processing on the GPU [9-10]. One work group includes a number of invocations of a compute shader. The user-defined compute space is defined by the number of work groups and the number of invocations within a workgroup. The work group space can be divided into three dimensions: X, Y, Z. The numbers of work groups in X and Y directions can be 1,024 at most, respectively, and the number of work groups in Z direction can be 64 at most. The whole three-dimensional work group space which includes all shader invocations is called the global compute space, and the space that includes the invocations within a particular work group is called the local work group space. The global work group is invoked into the compute shader using the glDispatchCompute() method, and the global work group that is dispatched is divided into local work groups. Since the invocations of the compute shader operate in a non-sequential manner and in parallel, it is important to properly divide the workload to the invocations of the compute shader in a sequence-independent structure. Fig. 2 shows the an one-dimensional global compute space and the local work group space within a particular work group.

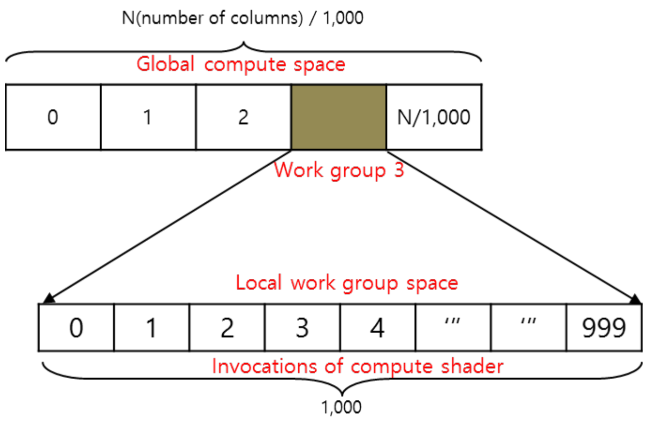

Fig. 2. One-dimensional global compute space and local work group space.

GLSL provides a shader sorage buffer object (SSBO) that is a data buffer accessible by CPUs and shaders. Uniform buffer objects (UBOs) used in existing shaders can store data up to 16KB, while SSBO can store data up to 128MB. In addition, there is no restriction on the data format that can be stored, and thus various operations can be performed. SSBO can be bound to a compute shader, making it easy to input and output dat needed for the task. In this paper, the data related to the mass-spring system and simulation of deformabled objects are stored in these SSBO. Based on the data stored in the SSBO, compute shaders for mass-spring system are invoked in parallel.

## 2.2. Mass-Spring System

The mass-spring system has a structure in which nodes with mass are connected to virtual massless springs. The representative types of springs are classified into shear, flex (bend) and structural springs depending on the connected structure. The mass-spring model is a structure in which the force applied to a node is propagated to other node through the springs connected to the node. Fig. 3 shows the different types of springs to be used in mass-spring system.

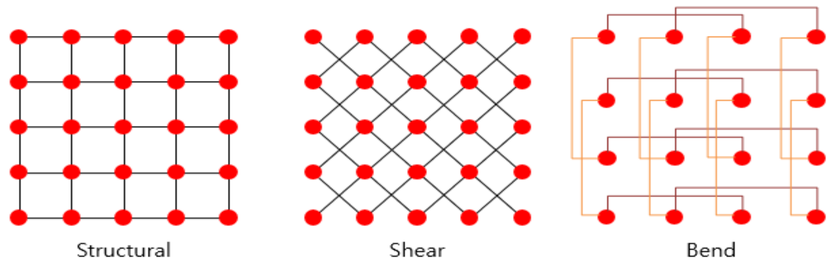

Fig. 3. Three types of springs.

The spring force transmitted to the two nodes constituting the spring is defined by spring coefficient (Ks), restoration coefficient (Kd), spring initial length (L0), and positions of two connected nodes (Node1, Node2). The calculated force values are used to obtain the next positions of two nodes through numerical integration. Numerical integration is a method of deriving a state value after a certain time interval by using an integration. There are several numerical integration methods such as such as Euler Integration, Midpoint Integration, and 4th Order Runge-Kutta Integration. Each scheme uses a unique approach in the iteration of time intervals, so the number of operations varies. Numerical integration methods accumulate numerical errors by proceed of time interval. Even though Euler method can accumulate numerical errors quickly, it has the least computational burden so that many applications requiring real-time processing have used it with small time interval.

## 2.3. Tetrahedron Volume Structure

Tetrahedron volume structure is a structure in which a volumetric object is divided into regular tetrahedron by adding vertices inside the existing surface mesh model. In the surface mesh model, the springs to support the inside does not exist so that the model can easily collapse when external forces are given. However, the tetrahedron mesh model has the advantage that it can preserve the shape of the model without the collapse since there are vertices and springs inside the model. Tetgen is a library for generating tetrahedron structural models and it can be used with 3d Max and Maya [11-12]. Standard file formats of surface mesh model such as obj and stl are used as inputs of Tetgen and it generates a tetrahedron model of the desired resolution as an output. In order to convert a surface mesh model into a tetrahedron model, Tetgen adds vertices inside the surface model by applying the delaunay triangulation algorithm based on the information of surface vertices. Based on the vertices generated finally, three kind of data, that is, vertex information, surface information, and the tetrahedron structure, are constructed for a volumetric object. **Fig. 4** shows examples of the tetrahedron mesh models converted from the existing surface mesh models by Tetgen.

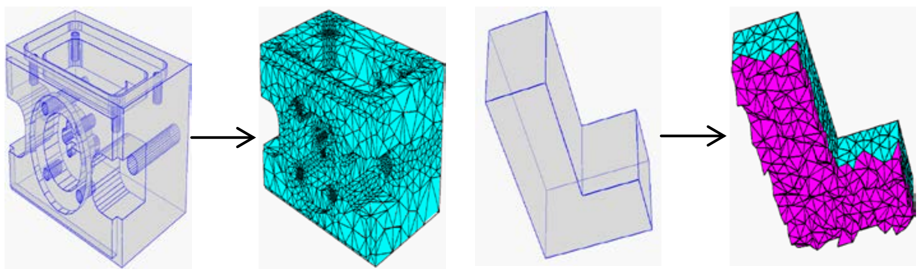

**Fig. 4.** Conversion from a surface mesh model to a tetrahedron model using Tetgen

## 2.4. Collision Detection

Collision detection is a major part of the physics-based simulation demaning high computational cost. A collision detection algorithm detects the collision of object and world, and the collision of object and object. Collision detection is classified into discrete method and continuous method. Discrete method is a method checking collision in every frame and continuous method checks collisoins between continuous frames. Continous method requires the more computation cost than discrete method because it checks collision based on object continuity. In order to detect collision effectively, the discrete collision detection methods have widely used by returning the positions of collided vertices to the prevlocation posistions and by giving collision responses, when the collisions are detected at one frame [13-14].

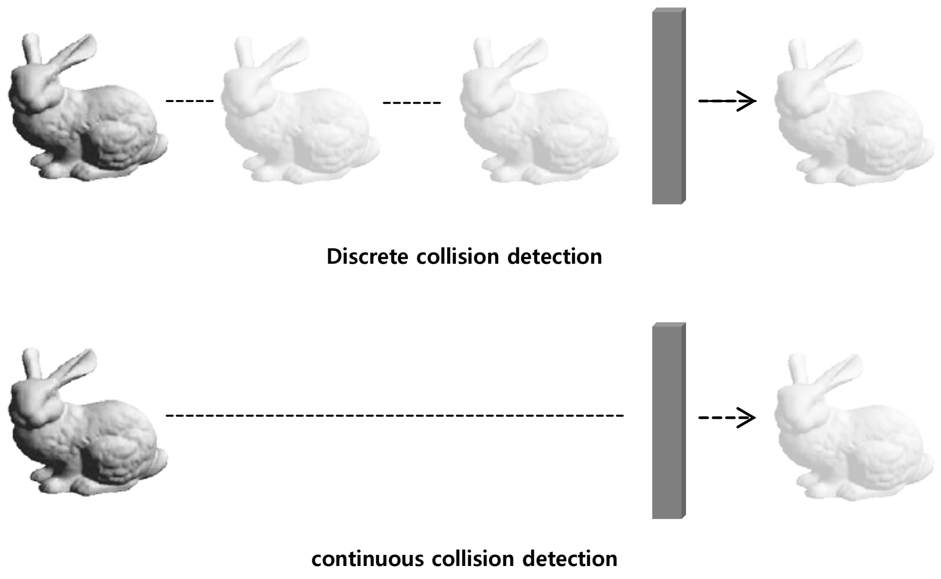

**Fig. 5.** Discrete collision detection and continuous collision detection

The collision detection method is largely classified into a method of detecting all nodes of an object and a boundary deteciton method of an object represented by a bounding volume. The method to check the collisions for every nodes in an object has increases the calculation overhead as the object becomes complicated and larger. In general, the boundary collision detection method has used as a culling step for choosing the colliding candidates before the accurate collision detection. The boundary is designated as a boundary larger than the object. The algorithm first checks collisions between objects by using boundary-based collision detection method. When a collision between boundaries is detected, a precise collision between the objects is detected using node-based collision detection method.

## 3. Design of Deformable Objects Simulation System

### 3.1. Compute Shader and SSBO Setup

The structure of the Compute Shader and the SSBO to construct the multi deformable object simulation system of this research is as follows. The Compute Shader is divided into two levels, that is Collision and Physics. The Collision Compute Shader level includes a Compute Shader that performs object boundary collision checking, surface collision checking, and subsequent collision response computation. Physics Compute Shader level includes Mass-Spring System, Light and Compute Shader that performs normal vector operations of objects. The structure of the Compute Shader specified in this research is shown in the following figure.

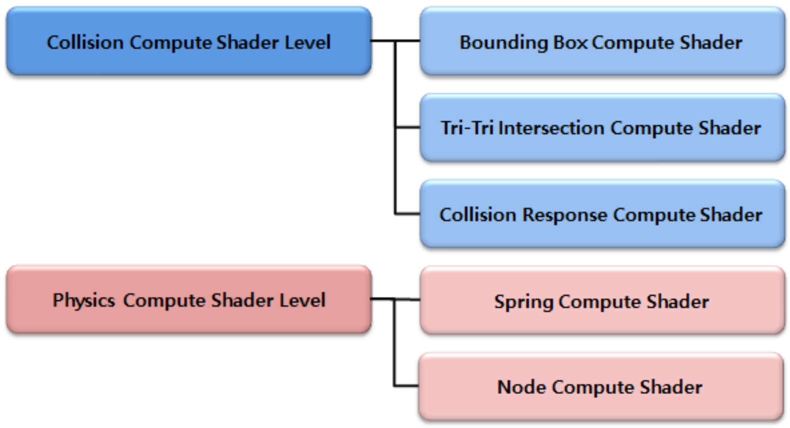
Collision Compute Shader Level
Bounding Box Compute Shader
Tri-Tri Intersection Compute Shader
Collision Response Compute Shader
Physics Compute Shader Level
Spring Compute Shader
Node Compute Shader

**Fig. 6.** Compute Shaders for our volumetric mass-spring system

The SSBO required for each Compute Shader and each component are shown in the following figure.

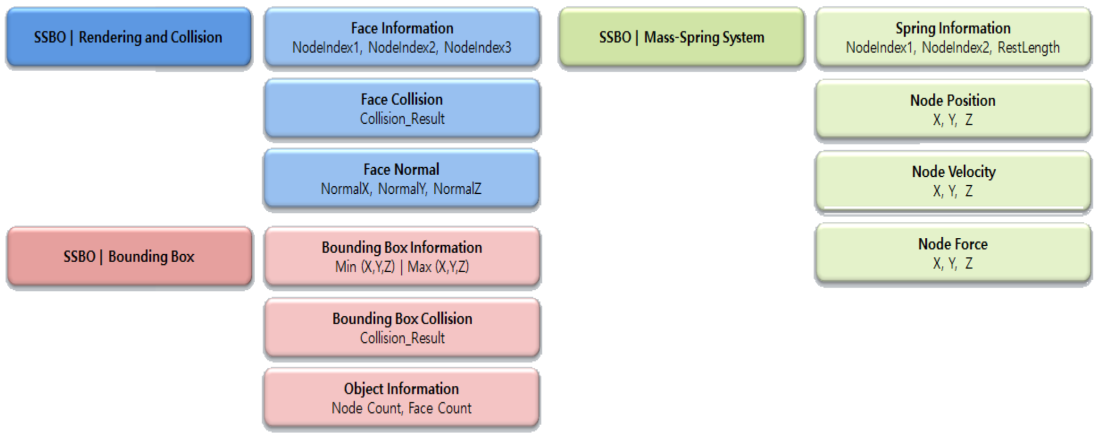

**Fig. 7.** SSBO structure map of system

There are seven SSBOs related to the object model: Node Position, Node Velocity, Node Force, Spring Information, Face Information, Face Collision Result, and Face Normal. Node Position stores the position information (x, y, z) of each node. Node Velocity and Node Force store velocity and force information (x, y, z) of each node. Spring Information SSBO stores the two node numbers that make up the spring and the initial length of the spring. These four SSBOs bind to the Physics Compute Shader Level, which performs physically based simulations. Face Informxation SSBO stores three surface nodes and is used for Face-Face Intersection and object rendering. Face Collision SSBO is a buffer that stores the results of Face-Face Intersection and is used for collision response. Face Normal SSBO is used to store Normal Vector for realistic object rendering. The bounding box SSBO for checking the primary collision stores information about the object number, six coordinate information forming the box, and the number of vertices and surfaces of the object.

## 3.2. Multiple Object's Data Mapping using SSBO

SSBO is generated by the order of Gen Buffer, Bind Buffer, Buffer Data Writing, and Bind Buffer Target. SSBO specifies the memory area in the Buffer Data Writing stage for effective memory allocation. After specifying the memory size of the whole SSBO, the data can be written by accessing the memory area directly through the rewriting process. In this research, we allocate the total memory size to bind Multi object to one SSBO, and then divide and write the memory area according to the size of each object.

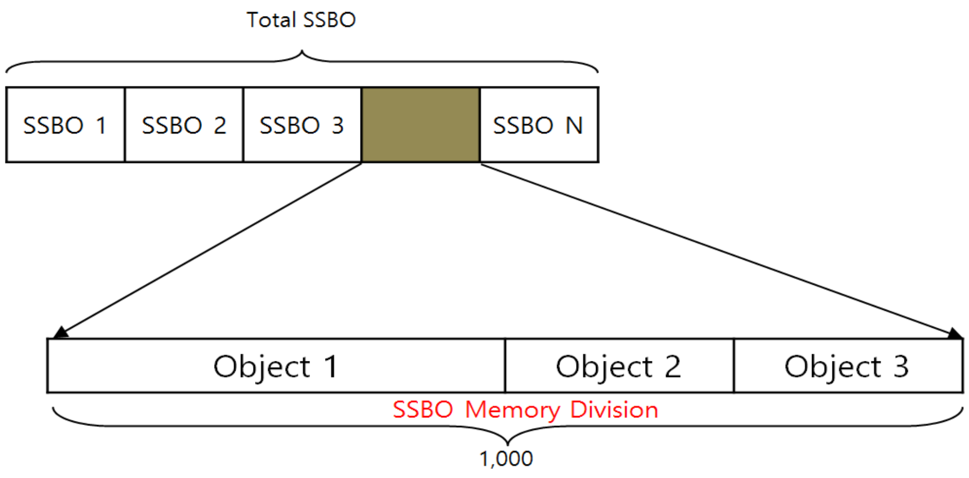

**Fig. 8.** SSBO memory allocation for multiple objects

## 3.3. GPU based mass-spring system design

The mass-spring system in the CPU calculates the force acting on the spring through the loop and then calculates the motion of the node. The following figure is a simple mass-spring system operation flowchart in CPU.

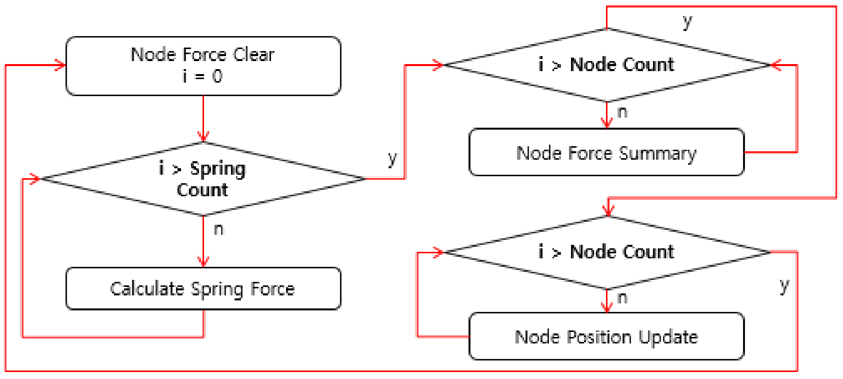

**Fig. 9.** Flow chart for CPU-based mass-spring system

The mass-spring system in the CPU uses 3 loops that computes the spring force for physics operations, a loop that adds up the node force, and a loop that updates the velocity and position by applying the force to the node. These three iterations are computationally fast if the object is simple and a single object. However, the complexity of the object, the slower the computation speed, the more realistic simulation can not be performed. However, GPU-based parallel processing allows the mass-spring system to perform operations on all objects simultaneously, which improves the simulation speed. Therefore, we construct a mass-spring system based on GPU-based parallel processing. The following figure shows a simple mass-spring system operation flowchart on the GPU.

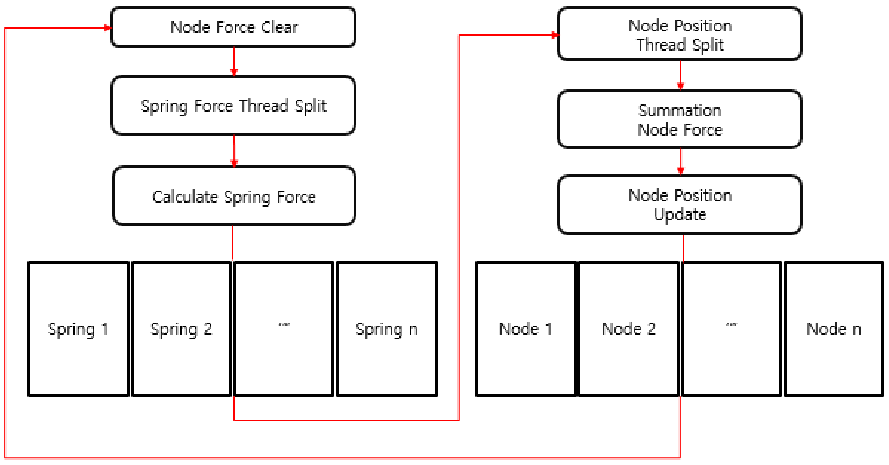

**Fig. 10.** Flow chart for GPU-based mass-spring system.

## 4. Implementation of Deformable Simulation System

### 4.1. Experimental Tetrahedron Mesh Models

In this research, deformable object simulation is performed by combining tetrahedron mesh model and Mass-Spring System. The modified object simulation system of this research consists of two phases, each using alphabet and old model. The model information used in the system is Node Position, Spring Information, and Face Information. The following table shows the information of each Tetrahedron model.

**Table 1.** Information of tetrahedron mesh models

| Tetra Model Name | Number of Node | Number of Spring | Number of Face |
|---|---|---|---|
| Sphere 1 | 4K | 23K | 8K |
| Sphere 2 | 1K | 11.3K | 2K |

The **Table 1** mean sphere models used in Phase 1 and 2. The phase 1 use three sphere1 models and phase 2 use sphere 1 and sphere2 models.

### 4.2. GPU-Based Mass-Spring System

In this research, we implemented the existing deformable object simulation system based on GPU using Compute Shader of GLSL 4.3. And it simulates multiple objects differently from existing simulation system. The simulation has two phases, simulating the alphabet freefall simulation and the crash simuluation of the sphere and sphere. The implementation hardware and software environment of the system are shown in the following table.

**Table 2.** Implementation and experimental environments

| Name | Specifications |
|---|---|
| CPU | Intel i7-3770K 3.5GHz |
| GPU | Nvidia GeForce GTX 760Ti |
| RAM | 32GB |
| OS | Windows 10 pro 64bit |
| IDE | Visual Studio 2013 Ultimate |
| Library | OpenGL 4.3, Tetgen |

The most important element of this modified object simulation system, GPU, was Nvidia GeForce GTX 760Ti. The GPU has a 915MHz Graphics Clock and a 980MHz Processor Clock. Both phases of the simulation system were run in the above implementation environment, and the experiment was performed in the same environment.

The following figure shows the multi deformable object simulation system implemented in this research.

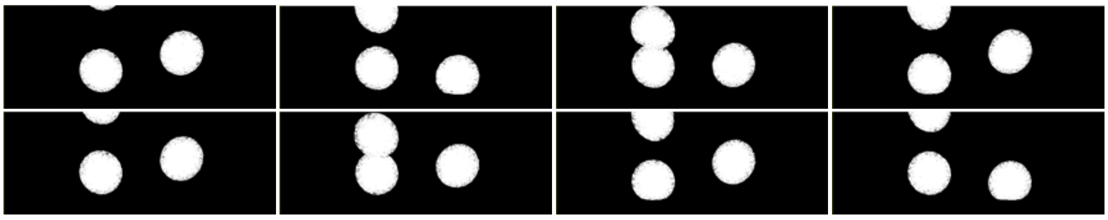

**Fig. 11.** Free fall simulation of multiple sphere models

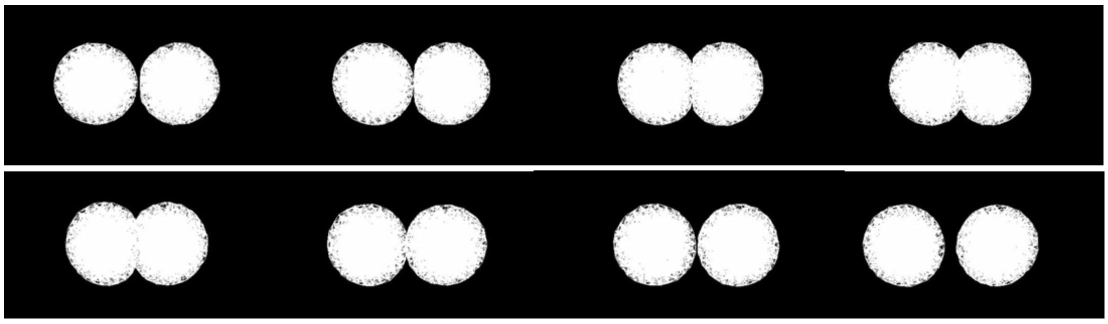

**Fig. 12.** Collision simulation of multiple sphere models

The alphabet simulation is a free fall of five alphabets and checks for collisions with other alphabets and floors. The sphere simulation moves to the same place and checks for collisions in a gravity-free situation. In this research, the system is implemented to simulate up to 5,000 frames. The maximum 5,000 frames we set in the system is the minimum frame condition in which objects fall and collide sufficiently.

## 4.3. Performance measurement experiment of simulation system

In this research, we compare the average frame per sec (FPS) to compare the simulation performance with the GPU in the CPU and the performance comparison with the bounding box. Simulation on CPU and GPU was performed by Face-Face Intersection after first collision check through the bounding box. The model used in the experiment was the same in CPU and GPU, and the average FPS was calculated 10 times in the same environments. The following table compares the average FPS on CPU and GPU.

**Table 3.** Experimental 1 : Performance comparison between CPU and GPU

| Simulation | FPS Average In CPU | FPS Average In GPU | Performance Improvements |
|---|---|---|---|
| FreeFall Multiple sphere models | 1.34 FPS | 131.66 FPS | 98.9% |
| Collision Multiple sphere models | 4.2 FPS | 218.11 FPS | 98.0% |

Experimental results show about 98 times performance improvement compared to CPU multi deformable object simulation. Although the CPU showed about 1.34, 4.2 FPS, the parallel simulation system with GPU shows 131.66, 218.11 FPS which exceeded 30 FPS, which is the minimum condition that does not give a sense of discomfort to the user.

Experiment 1 performs simulation including Bounding Box and Experiment 2 performs simulation that does not include Bounding Box in order to perform performance comparison according to presence or absence of bounding box in GPU. The experiment was performed 10 times in the same environment. Experiments are performed only until a collision occurs to compare the computation speeds associated with the collision tests before the collision.

**Table 4.** Experimental 2 : Performance comparison between without bounding box and with bounding box

| Simulation | Without Bounding Box | With Bounding Box | Performance Improvements |
|---|---|---|---|
| Free fall multiple sphere models | 39.11 FPS | 249.45 FPS | 84.3% |
| Collision multiple sphere models | 71.38 FPS | 343.82 FPS | 79.2% |

Experimental results show a performance improvement of more than about 80% over the case of continuous Face-Face Intersection without collision. By eliminating unnecessary collision check operations, the overall operation speed of the simulation system can be improved.

## 5. Conclusions

In this research we implemented multi deformable object simulation using OpenGL shader language, GLSL 4.3. We performed deformable object simulation that which requires many operations in parallel using GLSL 's Compute Shader. Our research showed a 98% to 99% performance improvement over the CPU-based simulation system using GPU parallel simulation system. We also used the bounding box method to efficiently perform collision detecting which takes up a large computational cost of the physics simulation system. It showed about 80% performance improvement compared to face-face intersection method which checks all faces of object. However, the one-level bounding box method that surrounds the outer surface of the object used in this study has a disadvantage in that it can not know the exact collision position of the object. This again increases the cost of the operation by using face-face intersection which checks all faces. In future works, it is expected that using hierarchical bounding box method will be able to find the exact collision position and optimize the operation for collision inspection more. It is expected that faster collision detection can be performed by applying the occlusion check algorithm using depth buffer.

## References

[1] M. Hong, J.H. Jeon, H.S. Yum, S.H. Lee, “Plausible mass-spring system using parallel computing on mobile devices,” *Human-centric Computing and Information Sciences*, vol. 6, no. 1, pp. 2016. Article(CrossRef Link)

[2] Vassilev, T. Ivanov, S. Bernhard, “A mass-spring model for real time deformable solids,” 2002. Article(CrossRefLink)

[3] J.H. Jeon, M.H. Choi, M. Hong, “Enhanced FFD-AABB Collision Algorithm for Deformable Objects,” *Journal of Information Processing Systems*, vol. 8, no. 4, pp. 713~720, 2012. Article (CrossRefLink)

[4] Selle, Andrew, Michael Lentine and Ronald Fedkiw, “A mass spring model for hair simulation,” *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3, 2008. Article(CrossRef Link)

[5] Adwait Jog, Onur Kayiran, Nachiappan Chidambaram, Onur Mutlu, Ravishankar Iyer, Chita R. Das, “OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance,” *ACM SIGPLAN Notices*, vol. 48, no. 4, 2013. Article(CrossRef Link)

[6] Campos, R. S, Lobosco, M, dos Santos, R. W, “A GPU-based heart simulator with mass-spring systems and cellular automaton,” *The Journal of Supercomputing 2014*, vol. 69, no. 1, pp. 1-8, 2014. Article(CrossRef Link)

[7] S. Collange, M. Daumas, D. Defour, D. Parello, “Barra: A parallel functional simulator for gpgpu. In Modeling,” in *Proc. of Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium*, pp. 351-360, Aug, 2010.
Article(CrossRef Link)

[8] Khronos Group, OpenGL 4.3, 2012. Article(CrossRef Link)

[9] D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane, “OpenGL programming guide: The Official guide to learning OpenGL, version 4.3,” 2013. Article(CrossRef Link)

[10] Khonos Group, OpenGL WIKI, "Compute Shader." Article(CrossRef Link)

[11] H. Si, "TetGen, a Delaunay-based quality tetrahedral mesh generator," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 2, 2015. Article(CrossRef Link)

[12] Tetgen, The Tetrahedron Mesh Model Generator. Article(CrossRef Link)

[13] S. Redon, A. Kheddar, S. Coquillart, "Fast Continuous Collision Detection between Rigid Bodies," *Computer graphics forum*, vol. 21, no. 3, pp. 279-287, 2002. Article(CrossRefLink)

[14] J. T. Klosowski, M. Held, J. S. Mitchell, H. Sowizral, K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-DOPs," *IEEE transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21-36, 1998. Article(CrossRef Link)

**Nak-Jun Sung** received the BS degrees in Computer Software Engineering from Soonchunhyaung University in 2016. Now he is undertaking a master degree of computer engineering courses as a member of the computer graphics lab at Soonchunhyang University. His research interests are computer graphics, GPGPU, VR (Virtual Reality) and AR (Augmented Reality)

**Min Hong** is an Professor at the Department of Computer Software and Engineering, Soonchunhyang University in Asan, Korea. He received BS in Computer Science from Soonchunhyang University in 1995. He also received MS in Computer Science and PhD in Bioinformatics from the University of Colorado in 2001 and 2005, respectively. His research interests are in Computer Graphics, Mobile Computing, Physically-based Modeling and Simulation, Bioinformatics Applications, and u-Healthcare Applications. In present, he is a Director of Computer Graphics Laboratory at Soonchunhyang University

**Seung-Hyun Lee** is a Associate Professor, School of Architectural Engineering, Hongik University. He received his B.S 1996 Inha University, and his M.S 1999 University of Colorado at Boulder, Ph.D. 2003 University of Florida at Gainesville 2004~2008:Korea Institute of Construction Engineering, Senior Researcher. 2008~Current: Hongik University, Associate Professor. Interesting Research Area: Process Simulation, Productivity, Construction Information Technology, Image Processing, Automatic Data Acquisition

**Yoo-Joo Choi** is a Professor in Department of Newmedia at Seoul Media Institute of Technology(SMIT), Korea. She received her M.S. and Ph.D. degrees in Computer Science from Ewha Womans University in 1991 and 2005, repectively. She was a researcher at R&D Department of KCI Co. and POSDATA Co. in Korea between 1991 and 1999. Her research interests include image processing, computer vision, computer graphics, augmented reality and human-computer interaction.