

유니티 게임 엔진에서의 구형 물체와 천 시뮬레이션간의 실시간 충돌 및 반응 처리 연구[☆]

REAL-TIME COLLISION RESPONSE BETWEEN CLOTH AND SPHERE OBJECT IN UNITY

김 민 상¹ 송 욱² 최 유 주³ 홍 민^{4*}
Min Sang Kim Wook Song Yoo-Joo Choi Min Hong

요 약

최근 컴퓨터 하드웨어의 성능이 증가함에 따라, 휴대용 전자 기기 뿐만 아니라 개인 컴퓨터에서도 더 사실적인 컴퓨터 그래픽 물체들을 생성하고 보여줄 수 있게 되었다. 이러한 이유로, 컴퓨터 그래픽을 포함한 디지털 콘텐츠는 더 계산적 비용이 높은 사실적인 가상의 물체들을 다양한 기기에서 실시간으로 표현하는 것을 요구한다. 멀티-플랫폼에서 구동되며 컴퓨터 그래픽을 포함한 게임, 애니메이션 등의 콘텐츠의 제작을 돕기 위해서는 유니티와 언리얼 엔진과 같은 기술들이 주로 사용된다. 시뮬레이션에서 더 사실적인 가상의 물체의 움직임을 표현하기 위해서는, 가상의 물체는 다른 물체들과 충돌해야 하며 현실세계와 비슷한 반응을 보여야 한다. 하지만, 다이나믹 시뮬레이션은 많은 계산 비용을 요구하나, 대부분의 휴대용 기기들을 이러한 다이나믹 시뮬레이션을 실시간으로 제공하지 못한다. 본 논문에서는 GPGPU 계산을 이용하여 구형 물체와 실시간으로 충돌 및 반응을 수행하는 천 시뮬레이션을 제안한다. 제안된 방법이 사실적인 디지털 콘텐츠에 유용할 것으로 기대된다.

☞ 주제어 : 천 시뮬레이션, GPGPU, 유니티, 다이나믹 시뮬레이션, 충돌 반응

ABSTRACT

As the performance of computer hardware has been increased in recent years, more realistic computer generated objects can be created and presented in personal computers and portable digital devices as well. For this reason, digital contents, including computer graphics, require virtual objects that are more realistic and representable in real-time on various devices, thus it requires more computational costs. In order to support the production of contents including computer graphics, games, and animations on multi-platform, Unity or unreal engines are mainly used. To represent more realistic behavior of virtual objects in a simulation, a virtual object must collide with other virtual objects and present the plausible interaction, as in the real world. However, such dynamic simulation requires a large amount of computational cost, and most portable devices cannot provide these dynamic simulations in real-time. In this paper, we proposed a GPGPU computation based dynamic cloth simulation to represent collision and response with spherical object in real-time. We believe that the proposed method can be useful for readily producing realistic digital contents

☞ keyword : Cloth simulation, GPGPU, Unity, Dynamic simulation, Collision-response

1. Introduction

Recently, as the optimization and integration of semiconductors have been improved, the performance of general digital devices has been increased dramatically, and the rate of improvement of processing speed of floating point, which is especially critical factor for expressing computer graphics, is improved at a remarkable rate. With the development of such computer hardware, personal

2017 and was selected as an outstanding paper.

^{1,2} Department of Computer Science, Soonchunhyang University, Asan, 31538, South Korea

³ Department of Newmedia, Seoul Media Institute of Technology, Seoul, 03925, South Korea

⁴ Department of Computer Software Engineering, Soonchunhyang University, Asan, 31538, South Korea

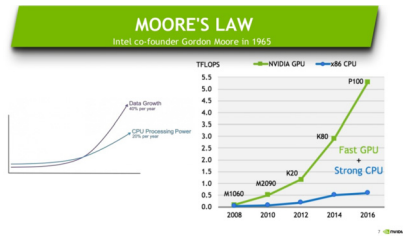
* corresponding author (mhong@sch.ac.kr)

[Received 27 August 2018, Reviewed 6 September 2018, Accepted 2 October 2018]

[☆] This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(Ministry of Science, ICT & Future Planning) (No. 2017R1A2B1005207).

[☆] A preliminary version of this paper was presented at APIC-IST

computers (PCs) and portable digital devices including smart phones have gained the ability to express the realistic computer graphics in real-time. Figure 1 compares the processing speed improvement between the GPU and the x86 CPU of NVIDIA, a typical GPU maker [1].



(Figure 1) The Comparing Amount of Performance Between x86 CPU and NVIDIA's GPU

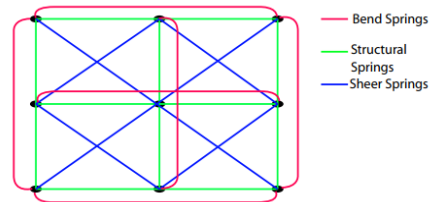
Along with the higher levels of computer hardware, computer graphics are evolving based on realistic representations that can interact with users in real-time on a variety of digital devices, including smart devices. In order to provide the same digital contents on various devices, there is a method of producing by programming for each platform and development using a multi-platform contents development tool which provides conversion to various platforms. In most cases, the latter is being used to develop the digital contents because of economic reasons and ease of maintenance [2]. Unity is the most representative multi-platform content development tool for most digital contents, including games, animations, and computer graphics, and it supports most digital devices, including smart devices.

The existing rigid-based computer graphics method has clear limitations on expressions such as cloth or deformable objects, so that most of these virtual objects in the real world are represented unrealistically. In general, digital contents should include dynamic simulations involving a variety of deformable objects, fabrics, and the like. However, this dynamic simulation requires a large amount of computational cost for expressing virtual objects and requires additional computational burden to calculate the interaction with other objects in real-time. In addition, for most smart devices, this real-time dynamic simulation cannot be provided due to performance limitations. In general, dynamic simulation

expresses an object in reality as a connection of particles containing discrete masses. The computation between each particle can be processed in parallel, and this parallel processing technique can provide real-time simulation even on mobile devices which have relatively low computational performance [3]. In this paper, we propose a method to perform various interactions between virtual cloth and spherical object in real-time using GPGPU in Unity, a multi-platform contents development tool.

2. Mass Spring System

Cloth simulation used in this paper is implemented in mass-spring structure. The mass-spring system is a traditional and simple method of expressing a deformable object using a point having a mass and a spring connecting the points, and can quickly present a deformable object in a computer simulation [4]. There are three types of spring in mass-spring system. Each spring type is as follows: a rectangular shaped structural spring that connects the adjacent mass-points on the top, bottom, left, and right sides of the cloth, a shear spring connecting the mass-points located diagonally, and bend spring that represents the resistance of the cloth when the mass-point is connected. Figure 2 shows the type of each spring [5].



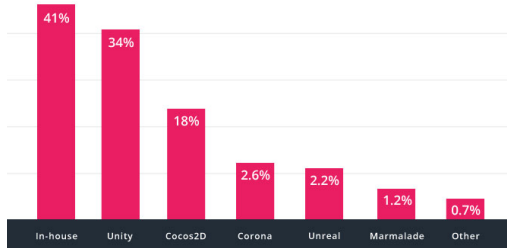
(Figure 2) Types of Springs Used in Mass Spring System

In the mass-spring system, externally applied forces are propagated through these springs and transmits forces to each connected mass-point by stretching and shrinking. At this moment the force F applied to a spring can be calculated by equation (1) where the position of connected two mass-points are p_i and p_j , spring constant is K , and the initial length of the spring is l . In this paper, we applied a simple square model for cloth simulation.

$$F = K(l - \|\overrightarrow{p_i + p_j}\|) \frac{\overrightarrow{p_i + p_j}}{\|\overrightarrow{p_i + p_j}\|} \quad (1)$$

3. Unity Engine

We use the Unity engine, which is a game engine that provides multi-platform conversion to apply the proposed cloth simulation to various hardware and operating system platforms. The Unity Engine is an integrated authoring tool for creating interactive contents such as 3D and 2D video games or architectural visualizations, with the highest share of the game engine market [6]. Figure 3 shows the market share of the game engine in the first quarter of 2016 and Unity engine has a market share of 34%. Unity Engine has the highest market share among single game engines [7]. Unity engine can hierarchically manage each game object displayed on the screen, and can add functions to each game object using javascript or script written in C # to control them.



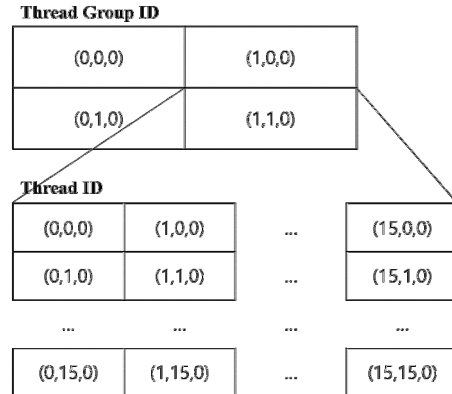
(Figure 3) 2016 Q1 Game Engine Market Share

3.1 Compute Shader in Unity

General-Purpose computing on Graphics Processing Units (GPGPU) was used to obtain the performance gain through the GPU for the parallel calculation of each mass-point on the mass-spring. GPGPU is a method to calculate outside the graphics pipeline area using the GPU. GPGPU cannot be recursively implemented, and its flow control performance is lower than that of the CPU, and the performance per core is somewhat lower than that of the CPU. However, since the number of processors is larger than that of the CPU and they provide the parallel computing for calculation, they perform

higher throughput than the CPU [8, 9]. In this paper, compute shader is applied to use GPGPU in Unity game engine.

Compute shaders in Unity are supported on computer hardware that supports shader model 5.0 or higher, or on platforms that support OpenGL 4.3 or higher, and are written in the High Level Shader Language (HLSL) language [10]. The compute shader is called from within a script written in the C # language, and is called in units of a three-dimensional thread group that corresponds to the parameter value passed in the call. Each thread group is internally made up of three-dimensional threads. In this paper, a thread group of size 16 x 16 x 1 is used in a configuration of 2 x 2 x 1 [11]. In this paper, the resolution of the cloth model used in the experiment is fixed to 32 * 32, which is considered by each thread to correspond to the mass-point of the cloth model. This is shown in Figure 4.



(Figure 4) Thread Construction in Compute Shader

4. Fourth Order Runge-Kutta

In this paper, the numerical integration is used to estimate the next state of velocity and position of discrete mass-points inside cloth model. To obtain a solution closer to the actual value, we used the fourth order Runge-Kutta method in this paper [12]. The fourth order Runge-Kutta has high accuracy among the numerical integration methods, but has a relatively high computational cost due to many computation steps. When there exists a differentiable curve $y = f(x)$

representing the mass-point velocity in the mass spring system, the fourth order Runge-Kutta method is calculated as follows. First, the value of the acceleration a_0 at $t=0$ is obtained as shown in equation (2).

$$a_0 = \Delta t F(x_n, y_n) \quad (2)$$

Using a_0 , the value of acceleration a_1 at $t_{\frac{1}{2}}$ is obtained as shown in equation (3).

$$a_1 = \Delta t F(x_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} a_0) \quad (3)$$

Using the obtained a_1 , the acceleration a_2 at t_1 is obtained as shown in equation (4).

$$a_2 = \Delta t F(x_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} a_1) \quad (4)$$

a_2 is used to obtain the acceleration a_3 at t_1 as shown in equation (5).

$$a_3 = \Delta t F(x_n + \Delta t, y_n + \Delta t a_2) \quad (5)$$

The final velocity of next time step can be obtained by summation of weight multiplied a_0 , a_1 , a_2 , a_3 added to current time step's velocity.

$$y_{n+1} = y_n + \frac{1}{6} \Delta t (a_0 + 2a_1 + 2a_2 + a_3) \quad (6)$$

The above process is repeated for every frame to estimate the velocity and position to obtain the next state of position for the mass-point for each frame.

5. Collision-Response in Simulation

5.1 Collision Detection

In dynamic simulation, which is performed in real-time, virtual objects are required to compute the collision detection

with other game objects and corresponding response should be reflected for realistic representation. In order to deal with the most basic collision model in 3D space, we have performed the collision between cloth and sphere type virtual object expressed in mass-spring structure in this paper. This collision-response update is performed after calculating the mass-point of the current frame from the mass-point in the previous frame, and before it is rendered. Since the mass-spring system in this paper is computed for each mass-point inside the GPU, we simply use the distance d between the mass-point and the sphere object to be collided with, and the radius r of the sphere object the collision processing operation can be performed simply. Figure 5 shows the pseudocode for simple collision detection.

```

1  Begin
2  distance = sqrt(pow((masspoint.x-sphere.x),2)
                    +pow((masspoint.y-sphere.y),2)
                    +pow((masspoint.z-sphere.z),2))
3  if(radius>distance)
4      return true
5  else
6      return false
7  End

```

(Figure 5) Pseudocode : Collision Detection with Sphere object

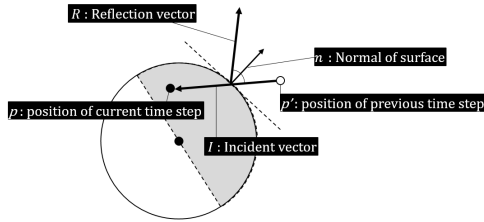
GPU-based shader programs have poor performance when they include branching operations such as pseudocode [13]. However, parallel execution is performed inside the compute shader because it has the performance advantage of offsetting it. After determining the collision through the above computation, the next collision response makes appropriate changes to each mass-point to enable natural interaction.

5.2 Collision Response

It is necessary to be recalculated the position of each

mass-point which is collided with the sphere object through the collision detection described above. In this paper, we applied two types of collision-response calculation methods and compare them. In the first traditional method, when the mass-point collides with the sphere, the velocity vector of the mass-point in the current frame is reflected on the spherical surface. The method of obtaining the reflection vector can be obtained as equation (7). As shown in Figure 6, the vector n applied in the formula to obtain the reflection vector is used by normalizing the vector from the center of the sphere to the mass-point located inside the sphere. In this case, it is assumed that the value of the delta time used in the numerical integration is sufficiently small that the position of the penetrating mass-point does not exceed the range of the hemisphere toward the position in the previous time step.

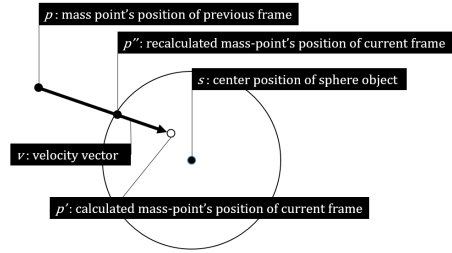
$$R = I - 2(N \cdot I)N \quad (7)$$



(Figure 6) Image of Vector Reflected by Sphere's Surface

The second method uses the ray-sphere collision to change the position of the mass-point. Since the coordinates of the mass-point at the moment of collision detection are located inside or on the surface of the sphere object, cast the ray from the position at the previous time step of the mass-point in the reverse direction of the velocity vector. Then move the position of the mass-point of the current time step to the point where the ray touches the spherical surface. This method also reflects a velocity vector on the spherical surface and it can be expressed as shown in Figure 7.

The coordinates of mass-point on the sphere surface satisfy the equation of the sphere and the equation of the straight line represented by the velocity vector, so it can be expressed as equation (8) and it can be expanded to obtain the



(Figure 7) Velocity as Ray Casted on Sphere Object

quadratic equation (9), simplified as $A^2 + 2B + C = 0$.

$$((p_x + tp_{dx}) - s_x)^2 + ((p_y + tp_{dy}) - s_y)^2 + ((p_z + tp_{dz}) - s_z)^2 - r^2 = 0 \quad (8)$$

$$(p_x - s_x)^2 + (p_y - s_y)^2 + (p_z - s_z)^2 - r^2 + 2t(p_{dx}(p_x - s_x) + p_{dy}(p_y - s_y) + p_{dz}(p_z - s_z)) + t^2(p_{dx}^2 + p_{dy}^2 + p_{dz}^2) = 0 \quad (9)$$

Applying a quadratic formula to the quadratic equation found in (9), we can find the time t until the moment the mass-point touches the surface of the sphere object from the previous frame. Since these calculations are made within the collision response branch after the collision detection operation is performed, the case where the solution of the quadratic equation does not exist is not considered.

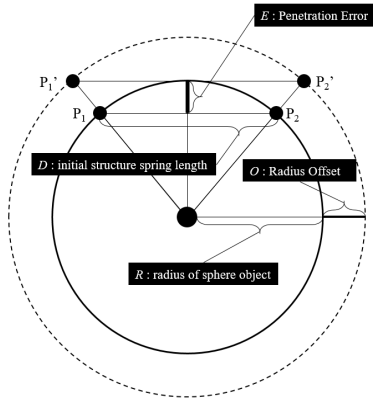
$$t_0, t_1 = -t((p_{dx}(p_x - s_x) + p_{dy}(p_y - s_y) + p_{dz}(p_z - s_z))) \pm \sqrt{(t((p_{dx}(p_x - s_x) + p_{dy}(p_y - s_y) + p_{dz}(p_z - s_z))))^2 - t^2(p_{dx}^2 + p_{dy}^2 + p_{dz}^2)}} \quad (10)$$

Since the two values t_0 and t_1 obtained through equation (10) must be positive. For the minimum errors as possible, the value of equation (11) is used for the calculation when the sign of equation (11) is positive, and value of equation (12) is used when the sign of equation (11) is negative.

$$t_0 = -t((p_{dx}(p_x - s_x) + p_{dy}(p_y - s_y) + p_{dz}(p_z - s_z))) - \sqrt{(t((p_{dx}(p_x - s_x) + p_{dy}(p_y - s_y) + p_{dz}(p_z - s_z))))^2 - t^2(p_{dx}^2 + p_{dy}^2 + p_{dz}^2)}} \quad (11)$$

$$t_1 = -t((p_{dx}(p_x - s_x) + p_{dy}(p_y - s_y) + p_{dz}(p_z - s_z))) + \sqrt{(t((p_{dx}(p_x - s_x) + p_{dy}(p_y - s_y) + p_{dz}(p_z - s_z)))^2 - t^2(p_{dx}^2 + p_{dy}^2 + p_{dz}^2))} \quad (12)$$

In this case, the surface of conflicting sphere object can penetrate the cloth because it is calculated for discontinuous mass-points. Therefore, we apply the offset value to the radius as shown in Figure 8 in the process of applying the collision response for more robust and realistic representation. The offset value can be obtained from equation (13).



(Figure 8) Radius Offset Calculated with Length of Structural Spring

$$O = \sqrt{\left(\frac{D}{2}\right)^2 + R^2} - R \quad (13)$$

Offset values must be calculated in pre-processing step, since the obtained offset must be applied to the collision-detection step in order to not pushing back mass-points position a lot. The radius offset was applied to the both suggested collision-response method. Since the radius offset values are mass-point resolution dependent, denser cloth model will represent more precise results.

6. Experiment

Experimental tests are performed under the computing environment as shown in Table 1. The parameters of the

cloth simulation are set as shown in Table 2.

In the simulation, only the two nodes located at both ends of one side of the cloth are fixed, and the simulation starts with 90 degrees rotated about the Z axis. Therefore, at the beginning of the simulation, the cloth model collides with the sphere object after it has fallen. The sphere object performs the reciprocating motion of the unit length back and forth with respect to the center of the cloth, and the velocity is slowly decelerated using the lerp function.

(Table 1) Environment SW and HW Specification of PC

Component	Specification
Operating System	Windows 10.0.16299.19
Unity	5.6.2f1 Personal
CPU	Intel(R) Core(TM) i5-4460
RAM	Samsung DDR3 PC3-12800 16GB
Mainboard	ASUSTeK H97M-E
Graphics Card	NVIDIA GeForce GTX 1070
VRAM	GDDR5 8GB

(Table 2) Parameters of Cloth Simulation

Parameter	Value
Cloth Size in Unity unit	1m x 1m
Cloth Mass	512
Spring Constant	35,000
Damping Constant	100
Cloth Mesh Resolution	32 x 32
Size of Thread group	16 x 16 x 1
Numbers of Thread group	2 x 2 x 1
Gravity Constant	-9.80665
Time Step	0.000005 per frame
Numbers of Spring	3,906

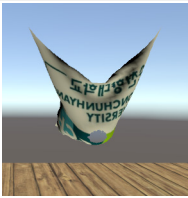
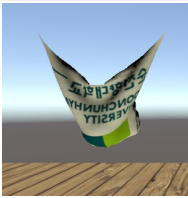
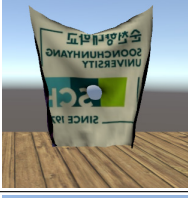
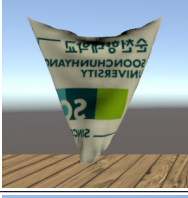


6.1 Collision-Response Experiments in Cloth Simulation

Two experimental tests are performed in this paper and they are as follows.

1. Comparison of the two proposed methods for the static sphere object after the collision-reaction
2. Comparison of two proposed methods for moving sphere object after collision-reaction
3. Comparing the performance between two proposed methods
4. Comparing the performance of second method implemented with CPU and GPU

First, the collision-response results between the static sphere object and the cloth were checked to compare the natural movement of the cloth and the phenomenon that the sphere surface penetrates through the cloth model's surface. Table 3 are snapshots of the 150, 500, and 800 frames in the simulation.

(Table 3) Performed Simulation with Static Sphere Object

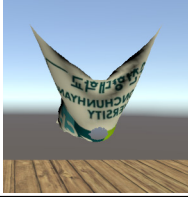
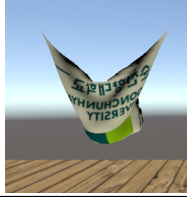

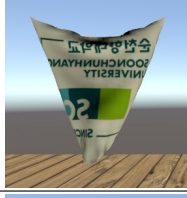


Frame number	First method	Second method
150		
500		
800		

In the first simple collision-response method, the face of the cloth model is penetrated in all three moments when the cloth falls on the simulation and touches the sphere object. On the other hand, the second proposed method shows that the face of the sphere object does not penetrate at all three

moments.

Table 4 compares the movements of the cloth model with moving virtual sphere object. In the 200 frame, the first time the cloth model collides with the sphere object, the second time the cloth model collides with the sphere object at 1,300 frame, and the third collision occurs at 3,100 frame. Frame 200 is captured from the back face of cloth model, because at that moment, back face shows the difference of two collision-response method more clearly.

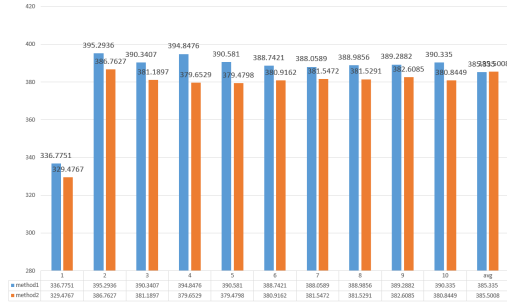
(Table 4) Performed Simulation with Moving Sphere Object

Frame number	First method	Second method
150		
500		
800		

In the case of a collision with a moving virtual sphere object, the edge of cloth is folded by the sphere object in the case of the first method in the 200 frame, while in the second method, the cloth model naturally surrounds the virtual sphere object. In the case of 1,300 frame and 3,100 frame, the first method is that the scale of velocity of the mass-point is not large enough so that the mass-point does not move sufficiently and the virtual sphere object penetrates the cloth model.

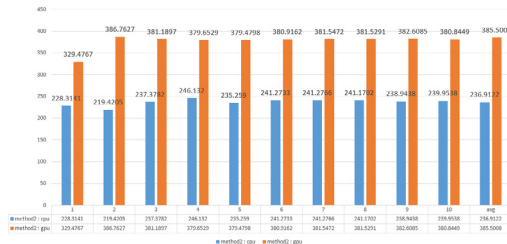
After comparing the visual performance, we compared the performance between two proposed methods. In this paper,

the performances were measured by averaging elapsed time between every 200 frames. The measurement was performed 10 times for each method.



(Figure 9) Performance Comparison between First Method and Second method

Since both methods show over 350 frames per second, we checked the benefit of GPGPU used in our implementation in Unity. In this comparison, the second proposed method was used, one calculate collision-response with GPGPU and the other with only CPU.



(Figure 10) Performance Comparison between CPU and GPU with Proposed Second Method

6.2 Experiment Result Analysis

Since the virtual sphere object in the cloth simulation has a discrete position value at every time step, a moment when a high velocity sphere penetrates the surface of the cloth may occur. Thus, the second method of moving the coordinates of the mass-point on the mass-spring structure to the sphere surface is better than expected to achieve a more natural and plausible results. And because the first method only changes the velocity vector, the sphere object penetrates the cloth when the cloth stops or moves at a slow speed. In addition,

the penetration does not occur in a low-resolution cloth model as it is used in experimental tests due to the effect of radius offset applied to collision-response.

In the first performance comparison, second method seems to be somewhat slower than first method, but still over 350 frames per second, so the both methods will be available on real-time simulations. In Figure 10, the first measured frames per second shows lower than other time because of the generating compute buffers and all initializations.

From the second performance comparison test, the implementation with GPGPU shows 60% faster than CPU-only calculations. Despite the second proposed method includes conditional branches and heavy calculations, the benefit of parallelism is bigger than cons as shown in Figure 10.

6. Conclusion

Recently, as the performance of computer hardware increases, it becomes possible to drive realistic and natural computer graphics on various devices. However, the behavior of objects that should be computed by dynamic simulation such as cloth collides with other objects on the simulation is difficult to be presented on the performance limit of the mobile hardware platform. Therefore, this paper proposed a more natural collision response method in the case of a mass-spring-based cloth simulation, and compared and analyzed them separately.

The result of moving the position of the mass-point at the collision-detected time step on the spherical surface is more natural than the method of reflecting only the velocity vector in the collision with the sphere object in the mass-spring structure. Based on this research, it is expected that it will be useful for the development of a cloth object which shows natural and plausible motion in real-time on digital contents targeting various hardware platforms.

References

- [1] Rommel Garcia, "GPU: The Beast In Data Centers", 2017 [Internet]
<https://www.slideshare.net/RommelGarcia2/gpu-101-t>

- he-beast-in-data-centers
- [2] Hartmann, Gustavo, Geoff Stead, and Asi DeGani. "Cross-platform mobile development." *Mobile Learning Environment*, Cambridge 16.9 (2011): 158-171.
 - [3] Cheng, H. (2009). *Interactive Cloth Simulation*. Science (pp. 1 - 8).
 - [4] Xavier provot, "Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior", *Graphics Interface*, pp.147-155, 1995.
 - [5] O'Connor, Corey, and Keith Stevens. "Modeling cloth using mass spring systems." *Appl. Soft. Comput* 12 (2003): 266-273.
 - [6] Unity, "Products", [Internet] <https://unity3d.com/unity>
 - [7] Unity, "Company Facts", 2017 [Internet] <https://unity3d.com/kr/public-relations>
 - [8] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., & Purcell, T. J. (2007, March). A survey of general purpose computation on graphics hardware. In *Computer graphics forum* (Vol. 26, No. 1, pp. 80-113). Blackwell Publishing Ltd.
 - [9] Boyd, Chas. "The DirectX 11 compute shader." *ACM SIGGRAPH*. Cité page 25 (2008).
 - [10] Unity Documentation, "Compute Shader", 2017, <https://docs.unity3d.com/Manual/ComputeShaders.html>
 - [11] MSDN, "numthreads", [INTERNET] [https://msdn.microsoft.com/en-us/library/windows/desktop/ff471442\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ff471442(v=vs.85).aspx)
 - [12] Davis, Philip J., and Philip Rabinowitz, "Methods of numerical integration," Courier Corporation, 2007.
 - [13] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). GPU computing. *Proceedings of the IEEE*, 96(5), 879-899.

● 저 자 소 개 ●



김 민 상(Min Sang Kim)

2017년 순천향대학교 컴퓨터소프트웨어공학과(공학사)

2018년~현재 순천향대학교 대학원 컴퓨터학과(공학석사)

관심분야 : 컴퓨터 그래픽스, 물리 기반 시뮬레이션, 안드로이드 애플리케이션

E-mail : ben399399@sch.ac.kr



송 욱(Wook Song)

2016년 순천향대학교 컴퓨터소프트웨어공학과(공학사)

2018년 순천향대학교 대학원 컴퓨터학과(공학석사)

관심분야 : Virtual Reality, Augmented Reality, Rehabilitation Contents

E-mail : wook2735@sch.ac.kr



최 유 주(Yoo-Joo Choi)

1989년 이화여대 전자계산학과(이학사)

1991년 이화여대 전자계산학과(이학석사)

2005년 이화여대 컴퓨터공학과(공학박사)

1991년 (주)한국컴퓨터 기술연구소 주임연구원

1994년 (주)포스테이타 기술연구소 주임연구원

2005년 서울벤처정보대학원 컴퓨터응용기술학과 조교수

2010년~현재 서울미디어대학원대학교 뉴미디어학부 부교수

2015년~현재 서울미디어대학원대학교 실감미디어연구소 교수

관심분야 : Computer Graphics, Computer Vision, HCI, Augmented Reality

E-mail : yjchoi@smit.ac.kr



홍 민(Min Hong)

1995년 순천향대학교 전산학과 학사(공학사)

2001년 University of Colorado at Boulder, U.S.A., Computer Science(공학석사)

2005년 University of Colorado at Denver, U.S.A., Ph.D in Bio Informatics(공학박사)

2006년~Present 순천향대학교 컴퓨터소프트웨어공학과 교수

관심분야 : Computer Graphics, Dynamic Simulation, Bio Informatics, Computer Vision

E-mail : mhong@sch.ac.kr