한국컴퓨터그래픽스학회
Korea Computer Graphics Society Vol. 25, No. 3, P. 55~63

질량스프링 시뮬레이션을 위한 병렬 구조 설계 방법

성낙준[1o] 최유주[2] 홍민[3*]

[1]순천향대학교 컴퓨터학과

[2]서울미디어대학원대학교 뉴미디어콘텐츠학부

[3]순천향대학교 컴퓨터소프트웨어공학과

njsung@sch.ac.kr, yjchoi@smit.ac.kr, mhong@sch.ac.kr

Parallel Structure Design Method for Mass Spring Simulation

Nak-Jun Sung[1o] Yoo-Joo Choi[2] Min Hong[3*]

[1]Department of Computer Science, Soonchunhyang University

[2]Department of Media Engineering, Seoul Media Institute of Technology

[3]Department of Computer Software Engineering, Soonchunhyang University

요 약

최근 물리 시뮬레이션 분야의 성능 개선을 위해 GPU 컴퓨팅 방식이 활용되고 있다. 특히 많은 연산의 양을 요구하는 변형물체 시뮬레이션의 경우 실시간성 보장을 위해 GPU 기반 병렬처리 알고리즘을 필요로 한다. 본 연구진은 변형물체 시뮬레이션을 구현하는 방법 중 하나인 질량스프링 시뮬레이션 기법의 성능을 향상시키기 위한 병렬 구조 설계 방법에 대한 연구를 수행하였다. 이를 위해 GPU에 직접 접근이 가능한 그래픽 라이브러리인 OpenGL의 GLSL을 사용하였으며, 독립적인 파이프라인인 컴퓨트 쉐이더를 활용해 GPGPU 환경을 구현하였다. 병렬 구조 설계 방법의 효과를 검증하기 위해 스프링 기반 질량스프링 시스템을 CPU기반과 GPU기반으로 구현하였으며, 실험의 결과 본 설계 방법을 적용하였을 때 CPU 환경에 비해 연산 속도가 약 6,000% 개선됨을 보였다. 추후 본 연구에서 제안한 설계 방법을 활용한다면 경량화 시뮬레이션 기술이 필요한 증강현실 및 가상현실 분야에 효과적으로 적용이 가능할 것으로 기대한다.

Abstract

Recently, the GPU computing method has been utilized to improve the performance of the physics simulation field. In particular, in the case of a deformed object simulation requiring a large amount of computation, a GPU-based parallel processing algorithm is required to guarantee real-time performance. We have studied the parallel structure design method to improve the performance of the mass spring simulation method which is one of the methods of implementing the deformation object simulation. We used OpenGL's GLSL, a graphics library that allows direct access to the GPU, and implemented the GPGPU environment using an independent pipeline, the compute shader. In order to verify the effectiveness of the parallel structure design method, the mass - spring system was implemented based on CPU and GPU. Experimental results show that the proposed method improves computation speed by about 6,000% compared to the CPU Environment. It is expected that the lightweight simulation technology can be effectively applied to the augmented reality and the virtual reality field by using the design method proposed later in this research.

*corresponding author: Min Hong/Soonchunhyang University(mhong@sch.ac.kr)

Received : 2019.06.08./ Review completed : 1st 2019.06.18. / Accepted : 2019.06.21.
DOI : 10.15701/kcgs.2019.25.3.55
ISSN : 1975-7883(Print)/2383-529X(Online)

키워드: 질량스프링 시뮬레이션, 병렬 처리, GPGPU

Keywords: MassSpring Simulation, Parallel Processing, GPGPU

1. 서론

최근 그래픽 하드웨어의 비약적인 발전과 더불어 다양한 GPU 컴퓨팅 아키텍쳐 및 알고리즘의 보급으로 인해 컴퓨터 그래픽스 분야 중 물리 시뮬레이션 분야의 성능 개선이 활발히 이루어지고 있는 상황이다.

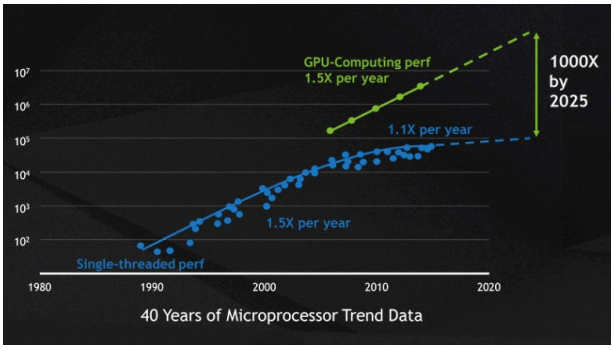


Figure 1: Computing Power Comparison between CPU and GPU [1]

물리 시뮬레이션은 힘을 가해도 모양이 변하지 않는 강체와 힘을 가했을 때 모양이 변하는 변형체로 구분할 수 있다. 강체를 시뮬레이션 할 경우 상대적으로 적은 연산을 수행하므로, 일반적인 CPU 기반 연산 구조에서도 실시간성을 보장할 수 있다. 하지만 변형체 시뮬레이션은 변형을 표현하기 위한 다양한 수학적 연산을 수행해야 하므로, CPU 기반 연산 구조에서는 실시간성을 보장할 수 없는 단점을 가진다.

변형물체 시뮬레이션을 위한 변형 연산은 주로 유한요소방법(FEM : Finite Element Method) 혹은 질량스프링 시스템(MSS : Mass Spring System)의 방법을 통해 구현된다. 유한요소방법은 변형을 처리하는 정확도를 중시하는 방법으로, 연산의 양이 많아 정확한 결과를 요구하는 건축·자동차 등의 분야에 활용된다[2-4]. 질량스프링 시스템 방법은 유한요소방법에 비해 연산의 양을 줄여 비교적 빠른 속도로 비슷한 정확도를 도출하는 데 의의를 두고 있으며 주로 옷감, 볼륨 물체 시뮬레이션에 활용된다[5-7]. 본 연구에서는 이러한 질량스프링 시스템 방법을 3D 볼륨 오브젝트 시뮬레이션에 적용해 실시간 시뮬레이션을 제공할 수 있도록 하는 병렬 구조 설계 방법에 대해 설명하고, CPU 기반의 시뮬레이션과 비교 실험을 수행한다.

본 논문의 2장에서는 GPGPU를 구성할 수 있는 아키텍쳐에 대한 연구 결과를 서술하고, 3장에서는 본 연구진이 병렬처리를 위해 사용한 GLSL의 Compute Shader 및 이를 위한 버퍼 구조인 SSBO에 대한 설계 방법에 대해 서술한다. 4장에서는 CPU 기반의 질량스프링 시뮬레이션과, 본 방법을 적용한 GPU 기반 질량스프링 시뮬레이션 방법의 성능을 비교한다.

2. 관련 연구

GPU의 높은 컴퓨팅 성능을 렌더링 목적 이외에 범용적인 용도로 사용할 수 있도록 하는 GPGPU(General Purpose Graphic Processing Unit)기술이 대두되며, 이를 손쉽게 사용하기 위해 다양한 병렬처리 아키텍처가 활용되고 있다. 기존 컴퓨터그래픽을 위한 라이브러리인 OpenGL의 GLSL, DirectX의 HLSL 등 고급 쉐이더 언어를 통해서 병렬처리를 수행할 수도 있지만, nVidia의 CUDA, OpenCL, DirectCompute 등 GPU를 이용한 범용 계산(GPGPU: General Purpose Computing on GPU)을 위하여 고안된 다양한 병렬처리 프레임워크를 사용하면 그래픽스 파이프라인의 이해 없이도 병렬처리를 손쉽게 구현할 수 있다[8-10]. OpenCL은 GLSL, HLSL과 같은 쉐이더 언어를 사용한 GPGPU 구성에 비해 구현이 효율적이며, 병렬 쓰레드간 동기화, 병렬화 알고리즘의 최적화 등에 효율적인 모습을 보인다. 그러나 그래픽 처리에 특화되어 있지 않기 때문에 화소 처리와 같은 작업을 수행할 때는 쉐이더 언어에 비해 비효율적인 모습을 보인다.

본 장에서는 다양한 병렬처리 아키텍처 가운데 그래픽처리에 원활한 모습을 보이는 OpenGL의 GLSL에 대해 서술하고, GPU 아키텍처를 활용해 물리 시뮬레이션을 구현한 연구를 분석한다.

2.1 OpenGL(GLSL)

대표적인 크로스플랫폼 그래픽 라이브러리인 OpenGL은 2012년 8월 OpenGL 버전 4.3을 발표하며 GPGPU를 위한 독립 파이프라인인 컴퓨트 쉐이더(Compute Shader)를 추가하였다[11]. 컴퓨트 쉐이더는 기존의 쉐이더가 수행하던 렌더링의 역할과는 다르게 GPGPU를 구현하기 위한 파이프라인을 제공하고 있어 직접적인 GPU 접근을 허용하고 있다는 장점을 가진다. 또한 기존에 사용하던 버퍼 객체인

UBO(Uniform Buffer Object), VAO(Vertex Array Object) 대신 쉐이더에 적합한 버퍼 객체인 SSBO(Shader Storage Buffer Object)가 새롭게 추가됐다. SSBO는 GPU의 V-RAM (Video RAM)에 따라 가변적으로 메모리를 사용할 수 있다는 특징 때문에 OpenGL 라이브러리에서 GPGPU를 구현하는 데 있어 컴퓨트 쉐이더와 함께 핵심적인 요소로 자리매김하였다. 그러나 컴퓨트 쉐이더의 연산 구조 및 SSBO의 설계 방법에 따라 GPU를 활용한 병렬처리 연산의 성능이 월등히 달라지므로 최적화된 구조로 컴퓨트 쉐이더와 SSBO를 설계하는 연구가 필요한 상황이다.

2.2 GPU 기반 변형물체 시뮬레이션 연구

최근 변형물체 시뮬레이션을 CPU환경과 GPU환경에서 각각 구현하고, 성능을 평가하는 연구가 많이 수행되고 있다. Huamin Wang, Yin Yang의 연구는 변형 물체 시뮬레이터의 성능을 향상시키기 위해 GPU 컴퓨팅 방법을 활용하여 단계 길이 조정, 초기화, 가역성 모델 변환 기술을 개발하였다. 이 때 메모리 효율성을 재고하기 위해 병렬처리 알고리즘을 설계하였으며, CPU를 사용했을 때에 비해 GPU를 사용하여 약 400배 이상의 성능 향상을 보였다[12].

Zhendong Wang et al.의 연구는 고해상도 천 시뮬레이션을 위한 최적화 방법을 제시하였으며 뉴턴의 방법 및 투영 다이내믹 방법에 대해 더 나은 성능을 보였다[13].

본 연구진의 선행 연구는 여러 개의 3D 구를 자유낙하하는 실험을 통해 CPU 환경과 GPU 환경의 시뮬레이션 성능을 평가하였으며, GPU 병렬처리 환경을 구성할 경우 약 84%의 성능이 향상되는 것을 보였다[14].

위와 같은 연구들이 진행이 되었으나 정확한 설계 방법을 제공하고 있는 연구는 진행이 되고 있지 않은 상황이다. 따라서 본 연구에서는 GPU 병렬성을 높이기 위한 버퍼 및 쉐이더 알고리즘 설계 방법을 제시하고, 이를 기반으로 성능을 평가한다.

3. 병렬 구조 설계 방법

본 장에서는 질량스프링 시스템을 위한 병렬 구조 설계 방법에 대해 설명한다. 먼저 질량스프링 시스템의 기본 구조에 대해 설명하고, 각 속성에 맞춘 버퍼 및 컴퓨트 쉐이더의 설계 방법에 대해 설명한다.

변형물체 시뮬레이션을 구성하는 방법 중 하나인 질량스프링 시스템은 물체를 질량을 가지는 점(노드)과 이를 연결하는 가상의 선(스프링)을 통해 구성하는 방식으로 다음과 같은 구조를 가진다.

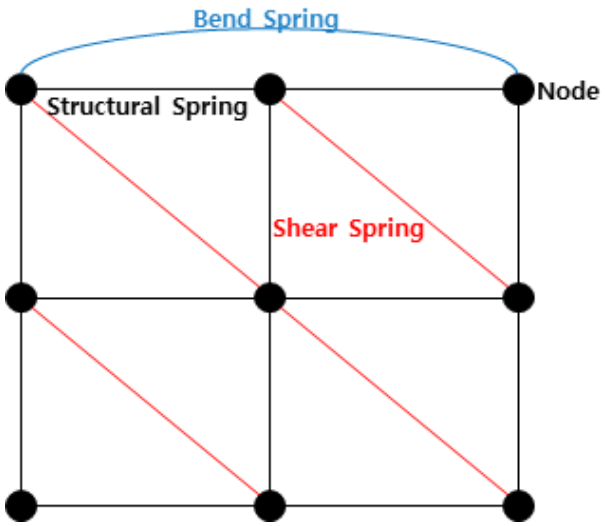


Figure 2: Mass Spring System Structure

크게 노드(Node)과 세 개의 스프링(Structural, Shear, Bend)로 구성되며, 각각의 구성 요소는 다음과 같은 데이터를 가진다. 노드는 각 점의 질량(Mass), 위치(Position) 및 속도(Velocity), 힘(Force) 기본 데이터로 가지며, 스프링은 연결된 두 노드의 번호(Index1 & Index2)와 연결된 초기 길이(RestLength), 그리고 스프링의 특성을 나타내는 스프링상수(Ks)와 댐핑 상수(Kd)를 기본 데이터로 가진다. 질량스프링 시스템은 이러한 노드와 스프링의 정보를 바탕으로 훅의 법칙(Hooke's Law)을 통해 스프링 힘을 계산해 변형을 표현한다. 스프링의 힘은 다양한 수치적분 방법을 통해 물체의 위치를 역추적하기 위한 매개체로 사용된다. 물체의 위치 예측을 통해 변형물체의 변형을 최종적으로 계산한다.

질량스프링 시스템의 기본 데이터는 3.1절 질량스프링 시스템을 위한 버퍼 설계 방법을 통해 GPGPU에 적합한 구조로 설계한다. 변형을 위한 훅의 법칙 및 수치적분 방법은 3.2절 질량스프링 시스템을 위한 컴퓨트 쉐이더 설계 방법을 통해 GPGPU에 적합한 구조로 설계한다. GPU의 병렬성을 높이기 위해서는 동시에 실행되는 컴퓨트 쉐이더 쓰레드의 작업 범위 설계과 쓰레드간 동시접근이 이루어지지 않는 것이 보장되는 SSBO의 설계가 중요하다.

질량스프링 시스템을 구현하는 방법은 크게 두 형태로 구분될 수 있다. 첫 번째는 노드 기반 구현방법이고 두 번째는 스프링 기반 구현 방법이다. 노드 기반 방법은 각 노드별로 쓰레드가 할당되어 노드에 연결된 모든 스프링에 대하여 for 문을 이용하여 접근해 스프링 힘을 계산하는 방법이다. 노드 기반 접근 방법에서는 같은 스프링에 대하여 두 번씩 계산이 되는 오버헤드가 발생한다, 그러나 스프링 기반 방법은 스프링별로 쓰레드가 할당되어 스프링 힘을 한번만 계산하기 때문에 노드 기반 방식에 비해 더욱 연산 속도가 빠르다. 그러나 스프링을 구성하는 노드에 스프링

힘이 누적될 수 있도록 병렬처리에 적합한 자료구조를 구축하여야 한다. 본 논문에서는 쉐이더 코드 안에서 for 문을 최대한 제거하고, 동일 대상에 대한 중복 계산량을 줄이기 위하여 스프링 기반 질량 스프링 구현 방법을 적용하고, 병렬성을 높이기 위한 버퍼 구축 방법을 3.1과 같이 설계하였다.

3.1 질량스프링 시스템을 위한 버퍼 설계 방법

SSBO는 OpenGL을 통한 GPGPU 환경을 구성하기 위해 사용하는 버퍼 오브젝트로, GLSL의 문법에서 지원하는 모든 데이터타입을 사용할 수 있는 것이 장점이다. 최적화된 GPGPU 환경을 만들기 위한 첫 번째 단계는 사용하고자 하는 데이터를 SSBO로 구성하는 것이다. 단순히 데이터를 로드해 버퍼에 저장하는 시스템의 경우 데이터 타입에 맞추어 SSBO를 생성하면 되지만, 연산을 통해 버퍼의 데이터를 변경하는 시스템의 경우 SSBO의 정교한 설계가 이루어지지 않으면 데이터의 충돌로 인해 잘못된 연산의 결과가 저장되는 경우가 발생한다.

다음 표는 질량스프링 시스템을 구성하는 노드와 스프링의 구조를 SSBO로 매핑한 결과를 나타낸다[15].

Table 1: SSBO Mapping result

노드		
데이터	**데이터타입**	**버퍼 이름**
질량(Mass)	float mass	-
위치(Position)	vector3 pos	SSBO Pos
속도(Velocity)	vector3 vel	SSBO Vel
힘(Force)	vector3 force	SSBO Force
스프링		
데이터	**데이터타입**	**버퍼 이름**
연결 노드1	int index1	SSBO Spring
연결 노드2	int index2	
연결 초기 길이	float RestLength	
스프링 상수	float Ks	
댐핑 상수	float Kd	

보통의 질량스프링 시스템에서의 모든 노드의 질량은 변하지 않고 같은 값을 가지는 상수로 설정한다. 버퍼에 이를 저장할 경우 메모리의 낭비 및 구조적 불편함을 유발한다. 따라서 질량스프링 시스템에서 서로 다른 질량으로 물체를 표현할 경우에만 SSBO에 질량을 포함시키는 것이 바람직하다. 또한 노드의 위치 및 속도는 시뮬레이션을 진행하며 매 프레임마다 결과가 변경되므로, 각각 별도의 SSBO로 구성하는 것이 바람직하다. 스프링의 경우 매 프레임마다 변경되는 데이터가 없으므로, 모든 스프링의 정보를 하나의 SSBO에 저장하는 것이 바람직하다. 스프링의 정보는 정수형 타입을 가지는 정보들과 실수형 타입을 가지는 정보가 혼재되어 있으므로, 일반적인 구조체 형태를 사용해 SSBO를 설계한다. 다음 그림은 각 버퍼의 메모리 구조를 도식화 한 것으로, 버퍼의 인덱스와 접근 방법을 나타낸다.

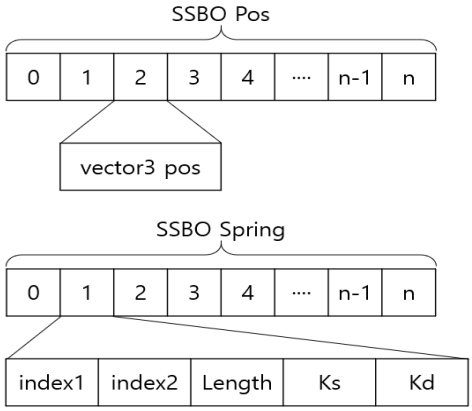


Figure 3: SSBO Structure

매핑된 SSBO는 위와 같은 형태로 메모리에 적재되며 배열과 같이 인덱스를 이용해 매핑된 데이터에 직접 접근이 가능하다.

3.2 질량스프링 시스템을 위한 컴퓨트 쉐이더 설계 방법

컴퓨트 쉐이더는 OpenGL 파이프라인과 독립적으로 수행되는 병렬처리를 위한 파이프라인으로, GPU에 직접적인 접근이 가능하다는 장점을 가진다. 컴퓨트 쉐이더는 처리할 작업을 x, y, z 3축 단위로 분할할 수 있으며, 이를 워크그룹이라고 한다. 워크그룹은 수행해야하는 작업을 쓰레드 형태로 분할하며, 쓰레드는 SSBO 단위의 입력과 출력을 수행한다. 워크그룹과 쓰레드의 형태는 다음 그림과 같다[16].

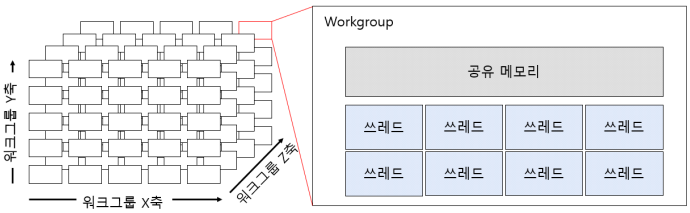


Figure 4: Work Group and Thread Structure

노드 기반의 질량스프링 시스템의 경우 노드의 정보만을 사용해 컴퓨트 쉐이더를 구성하기 때문에 하나의 컴퓨트 쉐이더를 통해 질량스프링 시스템을 구현할 수 있다. 그러나 쉐이더에서 for문을 사용해 스프링 힘을 계산하기 때문에 쓰레드의 연산 부하가 발생할 뿐 아니라, 두 번의 스프링 힘 계산이 수행되므로 불필요한 연산을 수행하게 된다. 이러한 점을 개선해 본 논문에서는 스프링 기반의 질량스프링 시스템을 설계하였으며, 스프링 기반 시스템의 컴퓨트 쉐이더 구성은 다음 그림과 같다. 스프링 힘을 계산하는 컴퓨트 쉐이더, 계산된 힘을 노드별로 합산하는 컴퓨트 쉐이더, 수치적분을 사용해 노드의 위치를 계산하는 컴퓨트 쉐이더를 반복해 질량스프링 모델을 매 프레임마다 업데이트하며, 업데이트가 된 이후 렌더링을 수행한다.

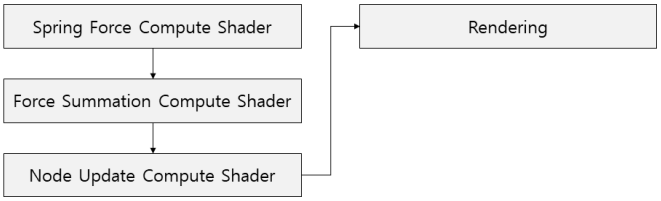


Figure 5: Compute Shader Flow

GPGPU 환경의 병렬 연산을 수행할 때, 결과를 저장하기 위해 SSBO의 인덱스에 동시접근(Conflict) 하는 상황이 발생할 경우 결과의 중복 저장으로 인해 잘못된 결과가 버퍼에 저장되는 문제가 발생한다. 따라서 동시접근 문제를 해결하기 위한 컴퓨트 쉐이더 설계 방법이 필요하다. 우리는 이를 해결하기 위한 방법을 설명하기 위해 연산 흐름 중 첫 번째 단계인 스프링 힘 계산 수식을 설명하고, 이를 직렬처리와 병렬처리 의사코드로 변환해 비교한다. 아래 수식은 스프링의 힘을 계산하기 위한 세 단계를 나타낸다.

$$Sitffness = (CurrLength - RestLength) * Ks \cdots\cdots\cdots\cdots\cdots\cdots (1)$$

$$Damping = \frac{VelocityDirection.(PositionDirection)}{CurrLength} * Kd \cdots\cdots (2)$$

$$SpringForce = (Sitffness + Damping) * \frac{PositionDirection}{CurrLength} \cdots (3)$$

수식(1)은 현재 스프링의 길이와 초기 스프링의 길이의 차를 통해 스프링의 강성을 계산한다. 수식(2)는 스프링을 이루고 있는 노드 간 속도의 차이와 위치의 차이를 통해 복원성을 계산한다. 이후 수식(3)을 통해 수식(1)의 결과와 수식(2)의 결과를 합쳐 스프링 힘을 계산한다. 이를 CPU에서 활용하는 직렬처리 알고리즘의 의사코드로 변환한 결과는 아래 표와 같다.

Table 2: Spring Force Calculate Algorithm

Input	Node(Pos, Vel), Spring
Output	Node(Force)
1	**BEGIN** for I=0, I<SpringCount, I++
2	Node1 = Node[Spring[I].index1] Node2 = Node[Spring[I].index2]
3	len = sqrt(PDir.x^2 + PDir.y^2 + PDir.z^2)
4	Stiffness = Ks * (len - RestLength) Damping = Dot(VDir, Pdir) / len * Kd
5	SpringForce = Stiffness + Damping SpringForce *= PDir / len
6	Node1.AddForce(SpringForce) Node2.AddForce(-SpringForce) **END**

직렬처리 방법에서 스프링 힘은 반복문을 통해 계산된다. 모든 스프링을 반복하며 스프링을 이루고있는 정보를 사용해 *Stiffness*와 *Damping*을 계산한 이후, 스프링을 이루고 있는 각 노드에 해당 힘을 더한다. 반복문을 사용하는 경우 순서대로 연산을 처리하기 때문에, 6번 문장에서 동일 메모리 접근 문제가 발생하지 않는다. 그러나 이를 병렬처리 방법으로 변경할 경우, 모든 쓰레드가 동시에 동작하기 때문에 6번 문장에서 동일 메모리 접근 문제가 발생한다. 병렬처리 시스템에서는 이를 해결하기 위해 연산이 끝난 뒤 모든 쓰레드가 각각 다른 메모리에 접근하도록 하는 방법을 사용한다. 저장을 수행하는 Output SSBO는 쓰레드의 연산의 결과를 동시에 저장할 수 있도록 Input1(*N*) * Input2(*M*)의 구조로 메모리 할당을 수행하고 각 쓰레드에 입력된 *N*과 *M*의 값에 의해 각 쓰레드가 사용할 인덱스(*Dst*)를 계산한다. 이 때 모든 쓰레드의 *Dst*는 중복되지 않음을 만족해야 한다.

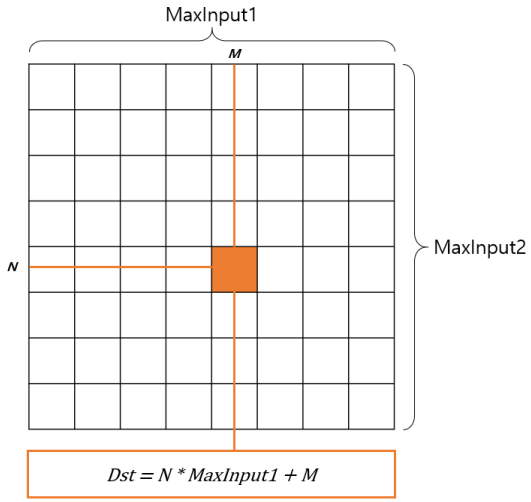


Figure 6: Output SSBO Structure and Dst Calculation Equation

위 그림은 스프링 힘 계산 알고리즘의 수행 결과를 저장하는 Output SSBO인 Force SSBO를 *MaxInput1* * *MaxInput2* 의 크기만큼 할당한 내용을 나타낸다. 또한 *N*과 *M*값에 따라 *Dst*의 위치를 계산하는 방법을 수식으로 나타낸다. 다음 표는 위 방법을 적용해 스프링 힘 계산 알고리즘을 병렬화 한 내용을 담고 있다.

Table 3: Spring Force Calculate Compute Shader

Input	SSBO Pos, Vel, Spring
Output	SSBO Force
1	**BEGIN** ID = gl_GlobalInvocationID.x Node1 = Spring[ID].index1

	Node2 = Spring[ID].index2 DstIndex1 = Node2 * MaxInput1 + Node1 DstIndex2 = Node1 * MaxInput1 + Node2
2	len = sqrt(PDir.x^2 + PDir.y^2 + PDir.z^2)
3	Stiffness = Ks * (len - RestLength) Damping = Dot(VDir, Pdir) / len * Kd
4	SpringForce = Stiffness + Damping SpringForce *= PDir / len
5	Force[DstIndex1] = SpringForce Force[DstIndex2] = -SpringForce **END**

병렬처리를 위해 각 쓰레드는 Node1의 번호(*N*)와 Node2의 번호(*M*)를 입력받는다. 해당 번호를 이용해 SSBO의 *Dst*를 계산한다. 이후 스프링 힘 계산 연산을 수행하고, 5번 문장에서 계산된 *Dst*를 사용해 계산된 스프링 힘을 저장한다.
이 때 저장된 스프링의 힘들은 각각 다른 공간에 저장됐기 때문에, 다음 시간의 노드 위치를 계산하기 위한 수치적분 연산에서 바로 사용할 수 없다. 따라서 이를 하나로 합산하기 위한 또 하나의 컴퓨트 쉐이더가 필요하다. 다음 표는 저장된 스프링 힘을 합산하기 위한 쉐이더를 나타낸다.

Table 4: Spring Force Summation Compute Shader

Input	SSBO Force
Output	vec3 ForceSum
1	**BEGIN** for I=0, I<NodeCount, I++ ForceSum += Force[I] **END**

병렬처리의 성능 저하는 반복문과 조건문에 의해 발생하지만, 버퍼의 다른 곳에 있는 모든 데이터를 하나로 합치기 위해서는 반복문의 사용이 불가피하므로 위와 같은 구조로 스프링 힘을 합산해 *ForceSum*이라는 하나의 버퍼에 저장한다. 이후 수치적분 연산을 수행하는 컴퓨트 쉐이더를 통해 합산된 힘을 통해 다음 노드의 위치를 계산한다.

Table 5: Node Position Update Compute Shader

Input	SSBO Pos, SSBO Vel, vec3 ForceSum
Output	SSBO Pos, SSBO Vel
1	**BEGIN** ID = gl_GlobalInvocationID.x vec3 veltemp = Vel[ID] vec3 postemp = Pos[ID]
2	acc = gravity + ForceSum/mass veltemp += acc * timestep postemp += veltemp * timestep
3	Pos[ID] = postemp Vel[ID] = veltemp **END**

위와 같은 형태로 컴퓨트 쉐이더를 구성하면 질량스프링 시스템을 시뮬레이션할 수 있으며 병렬처리에 적합한 구조로 직렬처리 방법에 비해 더욱 높은 성능을 도출할 수 있다.

4. 질량스프링 시스템의 구현

본 장에서는 3장에서 설명한 버퍼와 컴퓨트 쉐이더의 설계 방법을 적용해 질량스프링 시스템을 구현하고, CPU 기반의 직렬처리 시스템과의 비교를 수행한다. CPU 기반의 직렬처리 시스템과 본 방법을 적용한 GPU 환경의 질량스프링 시스템은 다음과 같은 환경에서 구현하였다.

Table 6: Development Environment

CPU	Intel i7 7700 3.60GHz : 8 Core
GPU	Nvidia GeForce GTX 1080Ti
RAM	32GB
V-RAM	11GB
Library	OpenGL GLSL 4.3
IDE	Visual Studio 2013 Ultimate

질량스프링 시스템을 통해 3D 볼륨 오브젝트를 생성하였으며, 변형을 처리하기 위해 Tetgen을 이용해 물체의 내부를 Tetrahedral 구조로 변형하여 사용하였다[17]. 실험에 사용된 3차원 구의 정보는 아래 표와 같으며, Wireframe 렌더링의 결과는 표의 그림과 같다.

Table 7: 3D Object Information

Number of Node	733
Number of Spring	23,004
Number of Face	7,926

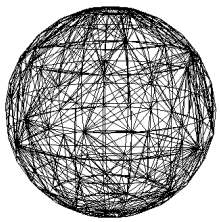
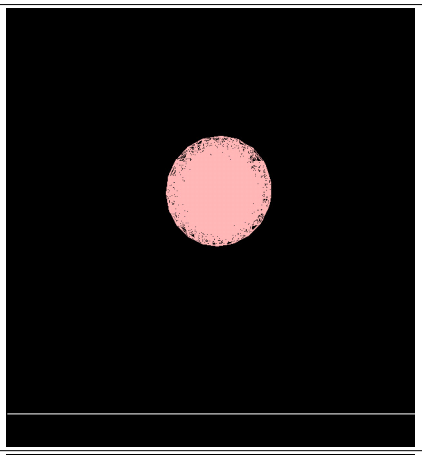
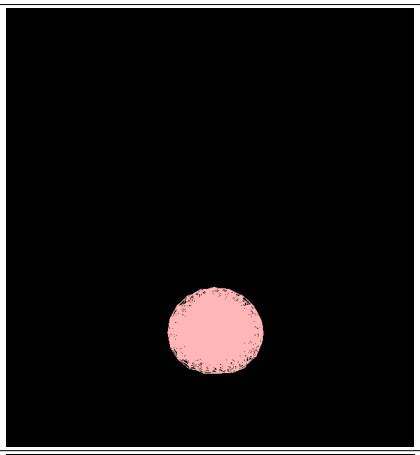
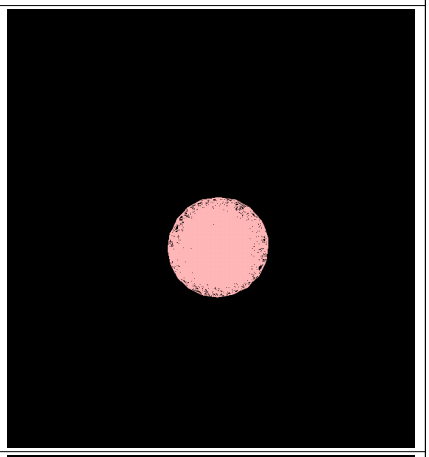
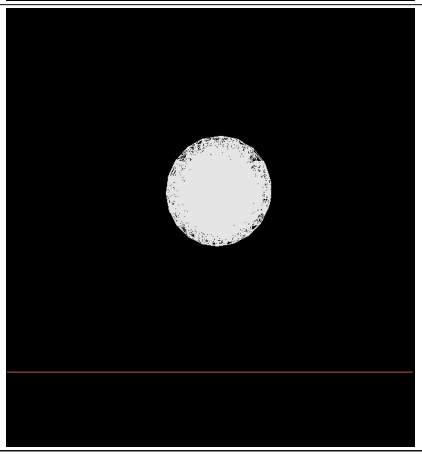
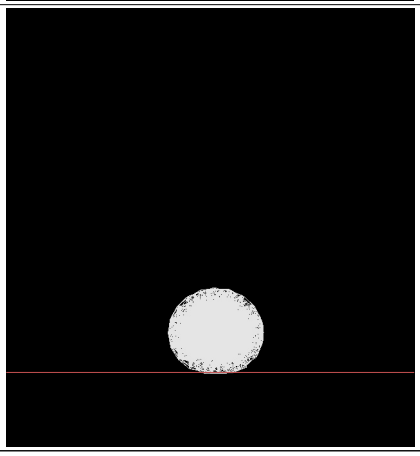
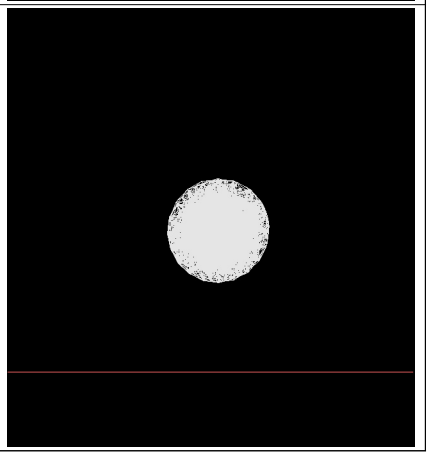


Figure 7: 3D Sphere

Table 8: Snapshots of Simulation(Before, In, After Collision State)

	Before Collision	In Collision	After Collision
CPU			
GPU			

본 논문에서는 병렬처리 구조에 따른 성능 분석에 초점을 맞추어 바닥과의 충돌처리만을 구현하였고, 추후 객체 간 충돌감지에 대한 병렬처리 구조를 설계 및 구현하고자 한다.
위 그림은 CPU와 GPU에서 구현한 변형물체 시뮬레이터를 바닥과의 충돌 이전, 충돌 상황, 충돌 이후 프레임에서 캡쳐한 결과를 나타낸 것으로, 연산의 결과를 검증하기 위해 동일한 움직임을 보이는 것을 확인하였다. 연산 성능을 비교하기 위해 프레임 당 평균 질량스프링 시스템 업데이트 연산 시간을 ms 단위로 측정하였으며, CPU 환경과 GPU 환경에서 모두 동일한 방식으로 연산을 수행하는 부분만 별도로 측정하였다. 다음 표는 CPU 환경과 GPU 환경에서의 프레임 당 평균 연산 수행 시간을 나타낸 것이다.

Table 9: Calculation Cost Comparison CPU and GPU
(단위: msec)

	1st	2nd	3rd	4th	5th
CPU	796	816	798	810	733
GPU	12	15	13	13	12
Improvement Ratio	6,533%	5,340%	6,038%	6,131%	6,008%

Table 9에서 성능개선률(Improvement Ratio)은 식(4)와 같이 구하였다.

$$ImprovementRatio = (CPUcost - GPUcost)/GPUcost * 100 \cdots (4)$$

CPU 환경에서는 1개의 변형물체를 시뮬레이션하는데 평균 790ms가 소요되었으나, GPU환경에서는 평균 13ms가 소요되었다. 약 6,000% 이상의 연산 속도 개선이 이루어졌으며, 하나의 물체를 시뮬레이션 할 때보다 여러 개의 변형물체를 시뮬레이션 하는 경우, 이러한 점이 더욱 극대화된다. CPU 환경에서는 최대 3개까지의 변형물체를 시뮬레이션할 때 30fps를 제공할 수 있지만, GPU환경에서는 최대 100개까지의 변형물체를 시뮬레이션해도 30fps 이상을 제공할 수 있다.

5. 결론

본 연구에서는 변형물체 시뮬레이션을 수행하기 위한 방법 중 하나인 질량스프링 시스템을 기반으로 병렬 구조가 물리 시뮬레이션 성능에 미치는 영향에 대해 비교하였다. 질량스프링 시스템을 구성하는 방법 중 반복문을 사용하지

않고 모든 쓰레드의 동시 작동을 보장할 수 있는 스프링 기반 방식으로 구현하였다. GPGPU 환경을 구성하기 위해 OpenGL의 컴퓨트 쉐이더를 사용하였으며, 스프링 기반 질량스프링 시뮬레이션에 적합하게 버퍼와 쉐이더를 구성하는 방법에 대해 서술하였다. 이를 기반으로 CPU 환경과 GPU 환경에서 실험을 수행한 결과 하나의 물체를 시뮬레이션 할 때에도 CPU 환경보다 GPU 환경에서 약 6,000%의 연산 성능이 개선됨을 확인하였으며, 많은 수의 변형물체를 실험할 때는 이러한 점이 더욱 극대화됨을 확인하였다. 본 연구에서는 쉐이더 언어인 GLSL을 사용해 병렬 시뮬레이션 시스템을 구성하였다. 이는 OpenGL을 적용할 수 있는 다양한 플랫폼에서의 구동이 가능함은 물론, 최근 다양화 되고 있는 WebGL을 사용한 시뮬레이션에도 활용이 가능하다는 장점을 가질 것으로 예상한다. 또한 시뮬레이션 성능만을 위한 병렬성 개선 자료구조 설계 및 실험을 수행하였으나, 충돌 검사 및 반응, 렌더링을 위한 노말 및 조명 처리 등 다양한 시뮬레이션 영역에서 병렬성을 극대화한다면 가상·증강현실 등 경량화 시뮬레이션 기술을 필요로 하는 다양한 분야에 접목이 가능할 것으로 예상한다.

감사의 글

본 연구는 순천향대학교의 연구비 지원 및 한국연구재단 이공학개인기초지원사업(NRF-2017R1D1A1B03035718)
의 지원에 의하여 수행된 연구입니다.

References

[1] Martin Heller, “What is CUDA? Parallel programming for GPUs”, Infoworld, 2018.

[2] Goldenthal, Harmon, Fattal, Bercovier, Grinspun, “Efficient simulation of inextensible cloth,” ACM Transactions on Graphics (TOG), vol. 26, no. 3, pp. 49, 2007.

[3] Nedel, Thalmann, “Real time muscle deformations using mass-spring systems,” In Proc of Computer Graphics International(IEEE), pp. 156-165, 1998.

[4] V. Baudet M. Beuve F. Jaillet B. Shariat F. Zara "Integrating Tensile Parameters in Hexahedral Mass-Spring System for Simulation" In Proc of 29th Int'l Conference Computer Graphics Visualization and Computer Vision (WSCG '09) pp. 145-152, 2009.

[5] Rahul Narain, Armin Samii, James F.O’brien, “Adaptive Anisotropic Remeshing for Cloth Simulation,” ACM transactions on graphics (TOG), vol. 31, no. 6, 2012.

[6] Oshita M., Makinouchi A., “Real-time cloth simulation with sparse particles and curved faces,” In Proc. of The Fourteenth Conference on Computer Animation of Computer Animation (IEEE), pp. 220-227, 2001.

[7] 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.

[8] Yang, Chao-Tung, Chih-Lin Huang, and Cheng-Fang Lin. "Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters." Computer Physics Communications vol. 182, no. 1, pp. 266-269, 2011.

[9] Ni, Tianyun. "Direct Compute: Bring GPU computing to the mainstream." GPU Technology Conference, pp.23, 2009.

[10] Stone, John E., David Gohara, and Guochun Shi. "OpenCL: A parallel programming standard for heterogeneous computing systems." Computing in science & engineering, vol. 12, no. 3, pp.66, 2010.

[11] Shreiner, Dave, et al. “OpenGL programming guide: The Official guide to learning OpenGL”, version 4.3. Addison-Wesley, 2013.

[12] Wang, Huamin, and Yin Yang. "Descent methods for elastic body simulation on the GPU.", ACM Transactions on Graphics (TOG), vol. 35, no. 6, 2016.

[13] Wang, Zhendong, et al. "Parallel Multigrid for Nonlinear Cloth Simulation.", Computer Graphics Forum, vol. 37. no. 7, pp.131-141, 2018.

[14] Sung, Nak-Jun, et al. "Simulation of Deformable Objects using GLSL 4.3.", KSII Transactions on Internet & Information Systems, vol. 11, no. 8, 2017.

[15] Kim, M., Sung, N. J., Kim, S. J., Choi, Y. J., & Hong, M. “Parallel cloth simulation with effective collision detection for interactive AR application”, Multimedia Tools and Applications, vol. 78, no. 4, pp. 4851-4868, 2019.

[16] Hong, M., Choi, Y. H., Sung, N. J., & Choi, Y. J., “Design and Implementation of Cloth Simulation Using GLSL 4.3”, Advanced Science Letters, vol. 23, no. 10, pp.10384-10389, 2017.

[17] H. Si, “TetGen, a Delaunay-based quality tetrahedral mesh generator”, ACM Transactions on Mathematical Software (TOMS), vol. 41, no. 2, 2015.

〈저자소개〉

성낙준

- 2012~2016 순천향대학교 컴퓨터소프트웨어공학과 학사
- 2016~2018 순천향대학교 컴퓨터학과 석사
- 2018~현재 순천향대학교 컴퓨터학과 박사과정
- 관심분야 : 변형물체 시뮬레이션, 병렬처리, 가상현실, 증강현실
- https://orcid.org/0000-0003-3514-7152

최유주

- 1989년 이화여대 전자계산학과(이학사)
- 1991년 이화여대 전자계산학과(이학석사)
- 2005년 이화여대 컴퓨터공학과(공학박사)
- 1991년 (주)한국컴퓨터 기술연구소 주임연구원
- 1994년 (주)포스데이타 기술연구소 주임연구원
- 2005년 서울벤처정보대학원 컴퓨터응용기술학과 조교수
- 2010년~현재 서울미디어대학원대학교 뉴미디어학부 부교수
- 2015년~현재 서울미디어대학원대학교 실감미디어연구소 교수
- 관심분야:컴퓨터그래픽스, 컴퓨터비전, HCI, 증강현실
- https://orcid.org/0000-0001-7520-097X

홍민

- 1995년 순천향대학교 전산학과(공학사)
- 2001년 University of Colorado at Boulder(공학석사)
- 2005년 University of Colorado at Denver(이학박사)
- 2006년~현재 순천향대학교 컴퓨터소프트웨어공학과 교수
- 관심분야 : 컴퓨터그래픽스, 다이나믹시뮬레이션, 바이오인포매틱스, 영상처리
- https://orcid.org/0000-0001-9963-5521