KIPS Tr. Software and Data Eng.
Vol.7, No.1 pp.25~32 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2018.7.1.25

GPU-Based Parallel Collision Detection for Deformable Objects

Nak-Jun Sung† · Kim Min Sang† · Min Hong†† · Yoo-Joo Choi†††

ABSTRACT

Due to heavy computational cost, deformable object simulation requires more effective collision detection method than rigid body simulation. However, when the CPU-based collision detection algorithm is purely applied to the GPU environment, the collision detection algorithm and the data structure optimized for the GPU environment are essential because the performance of the GPU can not be used properly. Therefore, we propose a GPU-based parallel collision detection algorithm for mass-spring system which is widely used for deformable object representation in this paper. The proposed method uses a parallel algorithm and data structure to reduce collision detection cost through GPU-based curling algorithm using AABB-Octree structure. In this paper, we prove the effectiveness of the proposed method by comparing the intersection test of all triangle pairs in parallel. The results of experimental tests show that the proposed method improves the performance by about 24% on average. Therefore, it is expected that the proposed method can improve the performance of real-time simulation for deformable objects.

Keywords : GPU, Parallel Processing, Collision Detection, Deformable Objects

변형 물체를 위한 GPU 기반 병렬 충돌 감지

성 낙 준† · 김 민 상† · 홍 민†† · 최 유 주†††

요 약

변형물체 시뮬레이션은 강체 시뮬레이션에 비해 많은 연산량을 요구하기 때문에 효과적인 충돌 검사 방법을 필요하다. 그러나 CPU 기반의 충돌 검사 알고리즘을 그대로 GPU 환경에 적용할 경우 GPU의 성능을 제대로 사용할 수 없기 때문에 GPU 환경에 최적화된 충돌 감지 알고리즘과 자료구조가 필요하다. 따라서 본 연구에서는 변형 물체 표현을 위해 널리 사용되고 있는 질량-스프링 시스템을 위한 GPU 기반의 병렬 충돌 감지 알고리즘을 제안한다. 제안하는 방법은 AABB-옥트리 구조를 이용한 GPU 기반의 컬링 알고리즘을 통해 충돌 감지 비용을 줄이는 병렬 알고리즘과 자료 구조를 사용하였다. 본 연구에서는 모든 삼각형 쌍의 충돌을 병렬로 검사하는 기존 방법과의 비교실험을 통하여 제안 알고리즘의 효율성을 입증하였다. 실험결과, 제안된 방법은 기존의 방법에 비해서 평균 약 24%의 성능 개선을 보였다. 따라서 제안하는 방법을 통해서 변형 물체에 대한 실시간 시뮬레이션의 성능 개선이 가능할 것으로 기대한다.

키워드 : GPU, 병렬 프로세싱, 충돌 감지, 변형 물체

1. 서 론

변형물체 시뮬레이션은 가상현실, 증강현실 등의 분야의 발전과 함께 높은 관심도를 보이고 있는 연구 분야 중 하나이다. 그러나 CPU기반의 단일 쓰레드 환경에서 변형물체 시뮬레이션을 구동할 경우 연산의 오버헤드가 극대화 된다. 최근 연구들은 변형 물체 시뮬레이션의 연산 속도를 증가시키기 위해 멀티 쓰레드 환경 혹은 GPU를 사용한 병렬 프로세싱을 통해 이러한 문제점들을 해결하고 있다.

Khronos Group에서 개발 및 배포하는 Open Graphics Library인 OpenGL도 2012년 OpenGL4.3을 발표하며 GPGPU에 최적화된 Compute Shader를 공개하였다[1]. OpenGL4.3에서는 기존에 사용하던 버퍼 객체인 UBO (Uniform Buffer Object), VAO(Vertex Array Object) 대신 SSBO(Shader Storage Buffer Object)가 새롭게 추가되었다. UBO와 VAO는 작은 용량의 메모리 한계가 존재하지만 SSBO는 GPU의 V-RAM(Video-RAM)에 따라 가변적으로 메모리 한계가 변하는 특징을 가지고 있다. 이는 고 해상도의 변형 물체를 시뮬레이션 할 수 있음을 의미하며, SSBO의 설계에 따라 시뮬레

※ 본 연구는 한국연구재단 이공학개인기초연구지원사업 기본연구지원사업 (NRF-2015R1D1A1A01059304)에 의하여 수행되었음.
† 준 회 원 : 순천향대학교 컴퓨터학과 석사과정
†† 종신회원 : 순천향대학교 컴퓨터소프트웨어공학과 교수
††† 종신회원 : 서울미디어대학원대학교 뉴미디어학부 부교수
Manuscript Received : November 14, 2017
Accepted : November 28, 2017
* Corresponding Author : Yoo-Joo Choi(yjchoi@smit.ac.kr)

이션의 정확도 및 성능이 달라짐을 의미한다. 그러나 GPGPU 환경을 구성하면서 일반적인 CPU 알고리즘을 적용할 경우 오히려 연산의 속도가 떨어지는 상황이 발생한다. 따라서 GPGPU 환경에 적합한 알고리즘의 설계가 필요한 상황이다.

변형 물체 시뮬레이션은 오브젝트 상태 업데이트, 충돌 검사, 충돌 반응, 렌더링 등으로 구분할 수 있다. 그 중 가장 많은 연산의 비용을 차지하는 부분은 충돌 검사 및 반응이다. 충돌 검사의 비용을 최적화하기 위해서 바운딩 박스(Bounding Box), 경계 구(Bounding Sphere) 등의 선행 연구가 수행되었다. 또한 이를 확장시킨 BVH(Bounding Volume Hierarchy), k-트리와 같은 공간 분할 기법(Spatial Division Method) 등의 방법이 고안됐다. 그러나 이러한 기법들도 GPU에 맞는 방식을 통해 수행되지 않으면 CPU방식보다 연산의 속도가 느리기 때문에, 최적화된 GPU 기반의 알고리즘의 구현이 필요한 상황이다. 따라서 본 연구에서는 GPU에 최적화된 충돌 감지 알고리즘과 더불어 변형 물체 시뮬레이션의 연산 속도 개선 방법에 관해 연구를 수행한다. 본 논문의 2장에서는 이와 관련된 연구들에 대한 논의를 수행하며, 3장에서는 GPU 기반 병렬 충돌 감지에 대한 알고리즘 설계 및 구현 결과에 관해 설명한다. 4장에서는 GPU 환경에서 충돌 시뮬레이션을 수행한 실험에 관한 결과를 설명하며, 5장에서는 결론 및 추후 연구사항에 관해 설명한다.

2. 관련 연구

2.1 변형물체 시뮬레이션

변형물체 시뮬레이션이란 물체에 힘을 가했을 때 모양의 변화가 일어나는 물체를 의미한다. 변형 물체 시뮬레이션을 표현하는 방법은 크게 연산 속도가 빨라 실시간 시뮬레이션에 적용이 가능한 질량-스프링 시스템(Mass-Spring System), 연산 속도는 느리지만 정밀도가 높아 정확한 표현이 필요할 때 사용하는 유한 요소 방법(Finite Element Method) 두 가지로 분류가 가능하다[2-4]. 질량-스프링 시스템(Mass-Spring System)은 물체를 이루는 정점(Vertex)과 그러한 정점 두 개로 이루어진 가상의 스프링(Spring)으로 구성된다. 변형 물체 시뮬레이션의 한 종류인 천 시뮬레이션(Cloth Simulation)에서는 Fig. 1과 같은 스프링 종류를 사용한다.

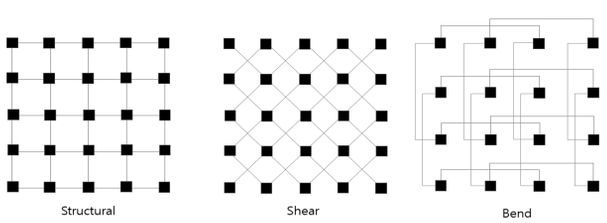


Fig. 1. The Different Kinds of Springs for Cloth Simulation

그러나 부피가 존재하는 3D 변형 물체들은 다양한 종류의 스프링을 사용하지 않고 구조 스프링(Structural Spring)만을 사용해 질량-스프링 시스템을 구성한다. 따라서 스프링의 다양한 정보를 표현하기가 어렵다. 이를 해결하기 위해서 물체의 스프링의 개수를 늘리는 방법을 사용한다. 그러나 CPU기반의 시뮬레이션은 물체의 스프링의 개수가 증가함에 따라 선형적으로 연산의 성능이 저하됨을 보인다[5, 6].

이전 연구에서는 선형적인 연산의 성능을 해결하기 위해 컴퓨트 쉐이더 구조를 사용한 병렬 연산을 사용하였다[4, 5]. 그러나 이러한 방법 또한 최적화된 병렬 알고리즘의 미흡함으로 인해 성능 개선율이 미흡한 상황이다.

2.2 질량-스프링 시스템이 적용 가능한 3D 모델의 생성

질량-스프링 시스템을 적용하기 위해서는 3D 물체가 표면만 있는 물체가 아닌, 내부에 정점과 스프링이 존재하는 물체여야 한다. 다양한 연구들에서는 이를 위해서 Tetgen을 이용한다[7]. Tetgen은 들로네 삼각분할 알고리즘(Delaunay Triangulation Method)을 활용해 표면 모델을 내부가 차있는 사면체분할 모델(Tetrahedral Volume Model)로 변경시키는 역할을 수행한다. 물체의 내부에 새로운 정점을 추가하는 과정을 통해 표면 개체의 수가 일부 증가하는 특징을 보인다. 다음 Fig. 2는 Tetgen을 이용해 표면 모델을 사면체 모델로 변환한 모습을 보여주고 있다.

Fig. 2. The Result of the Conversion from a Surface Model to a Tetrahedral Model by Tetgen (left) Surface Model (right) Tetrahedral Model

2.3 GPU 환경에서의 충돌검사 방법

과거부터 충돌 검사는 컴퓨터 그래픽스 연구 분야 중 가장 활발하게 연구가 이루어지고 있는 분야이다. 충돌 검사의 경우 물체에 힘을 가해도 모양이 변하지 않는 강체(Rigid-Body)보다 힘을 가하면 모양이 변하는 변형물체(Deformable-Body)일 때 더 많은 비용을 소모한다. 최근 연구들은 이러한 변형물체 시뮬레이션 상황에서 충돌 검사 연산의 비용을 줄이는 알고리즘에 관해 활발하게 수행되고 있다. 대표적으로 변형 물체의 충돌을 감지하는 방법은 경계 계층 구조(Bounding Volume Hierarchy), k-트리 공간 분할(k-Tree Spatial Division), 거리 필드(Distance Field) 등이 있다. BVH는 물체의 부피를 나눠 계층 구조로 정보를 저장해 충돌을 검사하는 방법으로 AABB(Axis Aligned Bounding Box), OBB(Object Oriented Bounding Box) 등과 함께 사용

된다[8-10]. k-트리 공간 분할은 물체가 존재하는 공간을 기준으로 특정 개수의 바운딩 박스로 물체를 분할하고, 해당 바운딩 박스들을 사용해 물체간의 충돌 혹은 자기자신과의 충돌을 검사한다[11-13]. 주로 이진트리(Binary Tree : 2), 쿼드트리(Quad-Tree : 4), 옥트리(OcTree : 8)로 공간을 분할해 충돌을 검사하며, AABB-옥트리가 주로 사용되고 있다. 다음 Fig. 3은 옥트리의 분할 과정을 나타낸다.

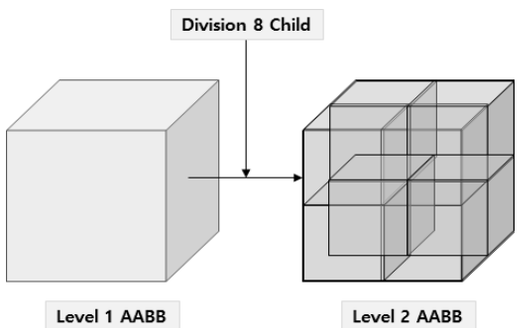


Fig. 3. Octree-Based Space Division

3. GPU 기반 병렬 충돌 감지

3.1 제안 방법의 개요

본 논문에서 제안하는 충돌 감지 방법의 파이프라인은 위 Fig. 4와 같다. 전처리 단계에서 오브젝트를 로딩하고 각 오브젝트의 정보를 계산 및 저장한다. 이후 SSBO와 컴퓨트 쉐이더(Compute Shader)를 생성한다. 오브젝트는 복셀에 의한 충돌 컬링을 거친 후 충돌이 일어난 복셀에 저장된 서로 다른 오브젝트간의 삼각형 집합간의 충돌 검사를 수행한다. 그리고 각 오브젝트의 렌더링을 수행한다.

3.2 전처리 단계

시뮬레이션을 위한 전처리 단계에서는 충돌 감지를 위해 SSBO와 컴퓨트 쉐이더를 생성한다. 본 논문에서 제안한 방법을 수행하기 위한 SSBO의 명세는 다음과 같다.

Table 1. Our SSBO Information

SSBO Index	Name
(0)	Node Position
(1)	Node Velocity
(2)	Spring Information
(3)	Node Force
(4)	BoundingBox Information
(5)	BoundingBox Collision Result
(6)	Face Normal
(7)	Face Information
(8)	Object Information
(9)	BoundingBox Face List
(10)	Face Pair
(11)	BoundingBox Mask

노드의 위치, 속도 정보는 SSBO(0), (1)에 저장되며 스프링을 구성하는 노드의 인덱스, 스프링의 초기 길이는 SSBO(2)에 저장한다. 질량-스프링 시스템의 업데이트 연산을 통해 구해지는 각 노드에 작용하는 힘은 SSBO(3)에 저장한다. 또한 복셀 기반 컬링을 수행하기 위해서 물체를 감싸는 바운딩 박스의 정보들을 SSBO(4), (5), (9)에 저장한다. 컴퓨트 쉐이더에서 효과적인 연산을 수행하기 위해 오브젝트의 부가적인 정보들을 SSBO[8]에 저장하여 사용한다. 다른 오브젝트간의 표면 삼각형 충돌 검사를 수행하기 위해서 복셀 기반 컬링의 결과를 저장하는 SSBO(11)과 표면 삼각형쌍의 정보를 담는 SSBO(10)를 사용한다. 각 SSBO의 데이터를 활용해 병렬 연산을 수행하는 컴퓨트 쉐이더의 명세는 다음과 같다.

Table 2. Compute Shader Information

Compute Shader Index	Name
(0)	BBUpdate.cshader
(1)	BBCollision.cshader
(2)	Triangle-Triangle Intersection.cshader
(3)	NodeUpdate.cshader
(4)	SpringUpdate.cshader
(5)	NodeForce.cshader

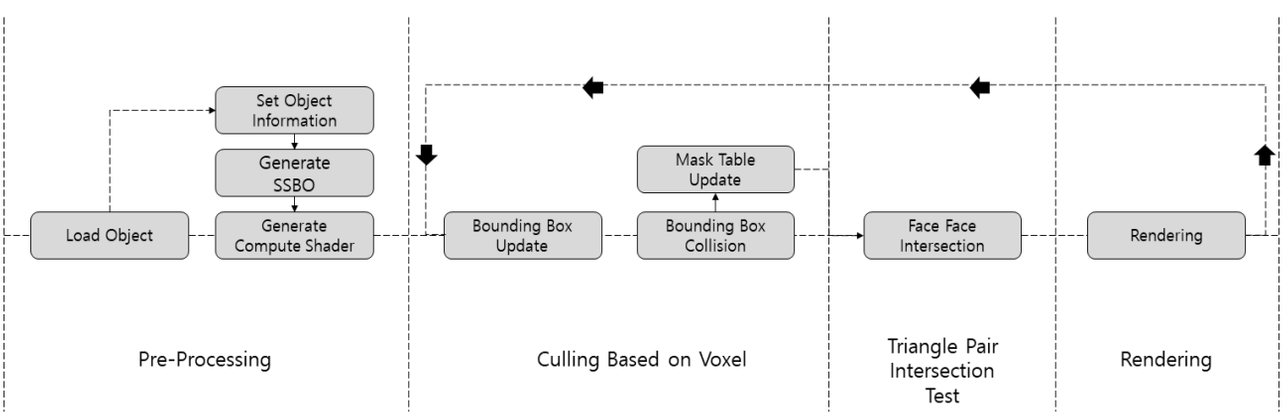


Fig. 4. The Pipeline of Proposed Method

컴퓨트 쉐이더(0), (1), (2)는 충돌 검사에 관련된 연산을 수행하는 컴퓨트 쉐이더이며, 컴퓨트 쉐이더(3), (4), (5)는 질량-스프링 시스템을 구성하는 데 사용되는 컴퓨트 쉐이더이다. 본 연구에서는 이러한 12개의 SSBO와 6개의 컴퓨트 쉐이더를 통해서 시뮬레이션 연산을 병렬로 수행한다. 또한 본 연구에서는 복셀 계층 구조를 사용하기 위해서 k-트리의 일종인 옥트리(Octree : 8-Tree)를 사용한다. 옥트리를 구성하는 정보는 AABB(Axis Aligned Bounding Box)를 사용하며, 전처리 과정에서 각 오브젝트의 바운딩 박스 정보를 초기화한다. 본 연구에서는 효과적인 복셀 기반 컬링을 수행하기 위해서 3-레벨 옥트리 구조를 사용하였으며, 해당 구조는 각 오브젝트마다 64개의 리프 노드(leaf node) 바운딩 박스를 가진다.

3.3 복셀 기반 컬링

복셀 기반의 컬링을 수행하기 위해서 가장 먼저 BBUpdate 컴퓨트 쉐이더를 호출해 각 박스가 가지는 표면 삼각형을 순회하며 바운딩 박스의 Min, Max 데이터를 업데이트한다. 업데이트가 종료된 이후 BBCollision 컴퓨트 쉐이더의 호출을 통해 서로 다른 오브젝트간의 바운딩 박스끼리의 충돌 검사를 수행하며, 이때 컴퓨트 쉐이더는 AABB 대 AABB의 충돌검사 알고리즘을 이용한다[14-16]. 각 컴퓨트 쉐이더의 의사코드는 다음과 같다.

Algorithm 1. BBUpdate Compute Shader

Input :	Node Position, BoundingBox Information, Face Information, Object Information, BoundingBox Face List
Output :	Updated BoundingBox Information
1	**Begin**
2	boxID = gl_GlobalInvocationID.x
3	Get Current Min, Max Value for boxID
4	for(all faces included in box(boxID))
5	{ now = each vertex for a face
6	if min.xyz>now.xyz, min = now
7	if max.xyz<now.xyz, max = now
8	}
9	
10	Set Min, Max value to Bounding Box Information [boxID]
11	
12	**End**

Algorithm 1의 BBUpdate 컴퓨트 쉐이더에서는 해당 바운딩 박스에 포함된 표면 삼각형들의 좌표에서 최소와 최대 좌표값을 찾아 Table 1의 (4)BoundingBox Information SSBO를 업데이트 한다.

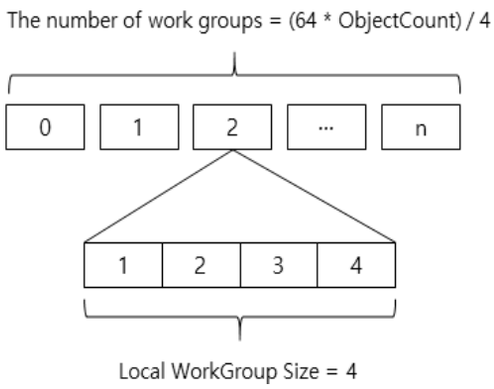


Fig. 5. Work Group Configuration for BBUpdate Compute Shader

BBUpdate 컴퓨트 쉐이더의 워크 그룹(Work Group) 구성은 Fig. 5와 같다. 워크 그룹의 수는 오브젝트의 수에 따라 가변적으로 변하지만 지역 워크 그룹 크기(Local Work Group Size)는 4로 고정하였다. 바운딩 박스간의 충돌을 검사하기 위한 BBCollision 컴퓨트 쉐이더의 의사 코드는 다음 Algorithm 2와 같다.

Algorithm 2. BBCollision Compute Shader

Input :	BoundingBox Information,
Output :	BoundingBox Mask
1	**Begin**
2	BB1 = gl_GlobalInvocationID.x;
3	BB2 = gl_GlobalInvocationID.y;
4	
5	//Initialize
6	Set Mask[BB1*64+BB2] = -1;
7	res =Check AABB_Collision[BB1,BB2]
8	
9	Set Mask[BB1*64+BB2] = res;
10	**End**

각 객체별로 64개의 AABB 바운딩 박스를 3.2절에서 설명한 방법에 따라 정의하고, 객체간 모든 AABB 바운딩 박스의 쌍에 대한 충돌검사를 BBCollision 컴퓨트 쉐이더에서 수행한다. BBCollision 컴퓨트 쉐이더에서는 바운딩 박스간 충돌 검사의 결과를 Table 1의 (11)BoundingBox Mask SSBO에 저장한다. Algorithm 2의 7번 줄은 Table 1의 (11)BoundingBox Mask SSBO의 인덱스와 두 객체의 바운딩 박스 인덱스 간의 상관관계를 나타내고 있다. BoundingBox Mask SSBO는 64 x 64의 2차원 구조정보를 저장하는 1차원 배열 구조이고, BoundingBox Mask SSBO의 행과 열은 각각 두 객체의 64개 바운딩 박스를 의미한다.

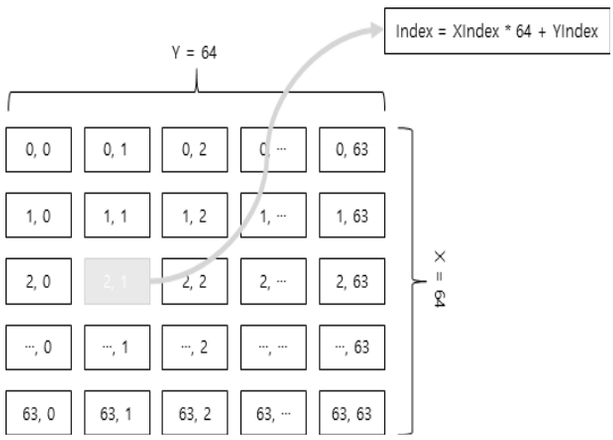


Fig. 6. Global Invocation Id for BBCollision Compute Shader

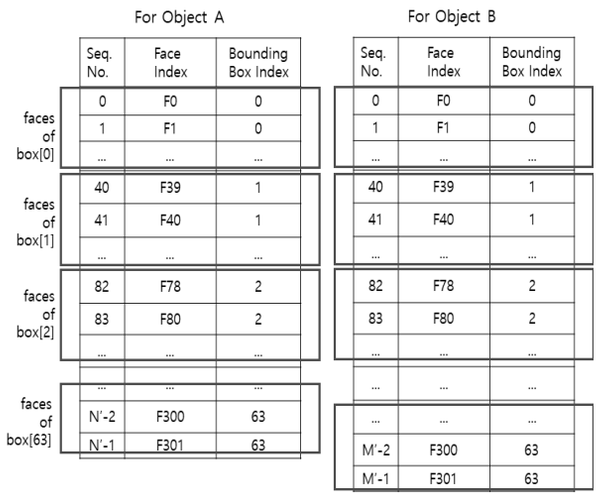


Fig. 7. Configuration of Bounding Box Face List SSBO

Fig. 6은 BoundingBox Mask SSBO의 index를 구하기 위한 행, 열의 인덱스를 표현함과 동시에 BBCollision 컴퓨트 쉐이더에 대한 Global invocation id를 나타낸다. 즉, 두 객체의 충돌감지를 위해 BBCollision 컴퓨트 쉐이더는 64 x 64 컴퓨트 공간(compute space)을 정의한다. 컴퓨트 쉐이더의 gl_GlobalInvocationID의 x와 y의 값을 통하여 충돌감지를 수행할 바운딩 박스의 인덱스와 충돌감지 결과를 저장할 BoundingBox Mask SSBO의 인덱스를 계산한다. 이와 같은 인덱스 공식을 통해 동시에 모든 바운딩 박스 쌍에 대한 충돌 검사를 수행하고 그 결과를 BoundingBox Mask SSBO에 저장할 수 있다. 이는 GPU 병렬 알고리즘을 통해 연산의 속도를 최적화 할 수 있는 가장 기본적인 형태이다.

3.4 표면 삼각형 쌍의 충돌 검사

3.3절에서 객체간 충돌 검사 성능을 개선하기 위해 복셀 기반의 컬링 방법[17-19]을 사용하였다. 즉, 충돌이 발생한 바운딩 박스들이 포함하고 있는 삼각형들의 쌍만을 선별하여 삼각형-삼각형 충돌검사를 수행한다. 삼각형 쌍의 충돌 검사를 수행하는 Table 2의 (2)Triangle-Triangle Intersection 컴퓨트 쉐이더는 전처리 단계에서 기구축한 Table 1의 (9) BoundingBox Face List SSBO의 크기에 비례하여 워크 그룹을 구축한다. Table 1의 (9) BoundingBox Face List SSBO는 64개 바운딩 박스에 포함된 삼각형의 리스트를 순차적으로 나열한 것이다. 이때, 각 행은 해당 삼각형에 대한 Table 1의 (7)Face information SSBO에서의 페이스 인덱스와 해당 바운딩 박스의 인덱스를 가진다. Fig. 7은 충돌 검색을 수행할 두 객체의 Bounding Box Face List의 예를 보여주고 있다.

객체 A를 위한 Table 1의 (9) BoundingBox Face List SSBO의 행의 수가 N'이고, 객체 B를 위한 행의 수가 M'이라하면, Table 2의 (2)Triangle-Triangle Intersection 컴퓨트 쉐이더를 위한 컴퓨트 공간은 N' × M'가 된다. 즉 병렬처리를 위하여 N' × M'의 쓰레드가 병렬로 구동된다.

Table 2의 (2)Triangle-Triangle Intersection 컴퓨트 쉐이더의 의사코드는 다음 Algorithm 3과 같다. Algorithm 3의 라인2-6에서 global invocation ID를 통하여 교차 테스트를 수행할 삼각형의 인덱스와 해당 삼각형이 포함된 바운딩 박스의 인덱스를 조회한다. 라인 9에서 바운딩 박스의 인덱스를 통해 Table 1의 (11)BoundingBox Mask SSBO에 접근해 두 삼각형을 포함하는 객체 A와 객체 B의 해당 바운딩 박스들이 충돌상황인지 그렇지 않은 상황인지를 체크한다. 바운딩 박스간 충돌이 이루어지지 않았다고 판단되면 리턴(return)하여 불필요한 연산의 수를 제거한다. 충돌이 된 상황이라면 라인 10에서 두 삼각형에 대한 세부 충돌 테스트를 수행하여 그 결과를 얻어오고 라인 11에서 테스트 결과를 Table 1의 (5)BoundingBox Collision Result SSBO에 저장한다.

Algorithm 3 : Triangle-Triangle Intersection Compute Shader

Input :	Node Position, BoundingBox information, BoundingBox Mask, Face Information, BoundingBox Face List
Output :	Updated Face-Face Collision Result
1	**Begin**
2	unint idx1 = gl_GlobalInvocatioID.x;
3	unint idx2 = gl_GlobalInvocatioID.y;
4	int BBindex1 = BBFaceList[idx1].bbindex;
5	int BBindex2 = BBFaceList[idx2].bbindex;
6	int Face1 = BBFaceList[idx1].faceindex; int Face2 = BBFaceList[idx2].faceindex;
7	// if box-box collision is happen,
8	// test triangle-triangle intersection
9	if (Mask[BBIndex1*64+BBIndex2] != 1) return;
10	Get Res = TriTriOverlapTest(Face1,Face2);
11	Set BoundingBox Collision Result(Res);
12	**End**

4. 실 험

4.1 실험 조건

본 연구에서 수행한 실험의 환경은 다음 Table 3과 같다. 병렬 충돌 감지 방법의 실험을 수행한 GPU는 V-RAM 8GB를 제공하는 GTX1070 모델을 사용하였으며, GPGPU 파이프라인을 구성하기 위해 OpenGL4.3(GLSL 4.3)을 사용하였다. 실험에 사용된 모델의 정보는 다음 Table 4와 같으며 모델들은 Tetgen을 통해 사면체 모델로 생성하였다.

Table 3. Experimental Environment

Name	Description
CPU	Intel i7-3770K
GPU	Nvidia GeForce GTX1070
RAM	16 GB
V-RAM	8 GB
OS	Windows 10
IDE	Visual Studio 2013 Ultimate
Library	OpenGL4.3

Table 4. Experimental Model Information

Model Name	A	B	D	L
Number of Node	2,320	1,369	1,418	4,233
Number of Tetra	7,867	4,702	5,399	17,889
Number of Face	3,724	2,220	2,020	5,156
Number of Spring	47,202	28,212	32,394	107,334
Number of Voxel	64	64	64	64

Tetgen을 통해 생성된 모델들을 다음 Table 5과 같은 Phase로 구분하여 충돌 시뮬레이션을 수행하도록 한다. Phase 1은 A와 L모델을 사용하며, Phase 2는 B와 D 모델을 사용한다.

Table 5. Experimental Phase Design

	Used Model Name	
Phase 1	A	L
Phase 2	B	D

4.2 실험 결과

본 연구에서 수행한 충돌 시뮬레이션은 초기에 두 물체 사이에 일정 거리를 두고 떨어져 있도록 위치시킨다. 이후 점차 상대 모델의 위치로 이동하며 교차하도록 설정하였다. 각 실험에서는 매 프레임마다 삼각형 교차 테스트를 병렬로 처리하는 일반적인 방법과 본 연구를 통해 제안한 방법을 이용해 동일한 상황을 시뮬레이션 하도록 한다. 이후 시뮬레이션에 소요되는 총 실행 시간을 측정해 일반적인 방법과 제안한 방법의 성능을 비교한다. 다음 Fig. 8과 Fig. 9는 실험 Phase 1과 실험 Phase 2의 충돌 전, 충돌 상태, 충돌 후의 모습을 나타낸다.

각각의 실험은 Wire-Frame상태로 렌더링 하여 결과를 출력하였으며, 충돌이 나타날 경우 충돌이 일어난 표면 쌍의 색상을 변경하도록 구성하였다. 보다 정확한 시간 측정을 수행하기 위해 충돌 전, 충돌 상태, 충돌 후의 프레임을 각각 1,500프레임 정도 수행되도록 시스템을 설정하였다.

Fig. 8. Phase 1 of Collision Simulation (left) Before Collision, (middle) in Collision, (right) After Collision

Fig. 9. Phase 2 of Collision Simulation (left) Before Collision, (middle) in Collision, (right) After Collision

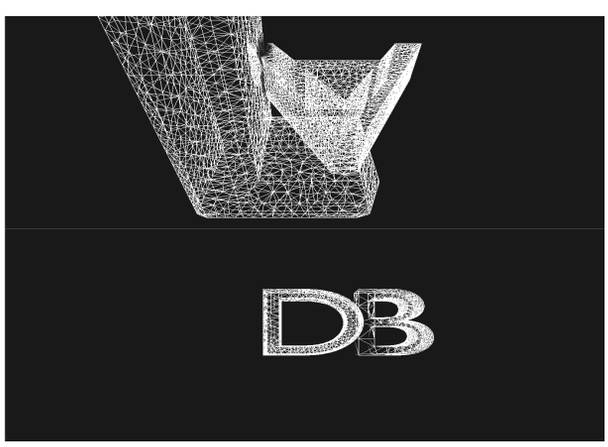


Fig. 10. Snapshots of Each Collision Simulation Phase

Fig. 10는 본 실험의 충돌 상황을 보다 자세하게 보여주고 있다. 충돌이 일어난 물체 1의 표면 색상을 color(1,0,0)로 칠하며, 물체 2의 표면 색상을 color(0,0,1)로 표현하였다. 이를 통해 본 시스템의 충돌 검사 연산이 원활하게 수행되고 있음을 가시적으로 확인 할 수 있었다.

실험의 결과는 수행 시간(초)로 도출되며 성능 개선율 I는 다음 식 (1)의 성능 개선율 공식을 사용하여 도출하였다.

$$I = (V_1 - V_2)/V_2 \quad (1)$$

여기서, V1과 V2는 각각 기존 방법과 제안 방법 적용 후 수행 시간을 의미한다.

다음 FIg. 11은 본 연구의 실험 결과를 나타낸다.

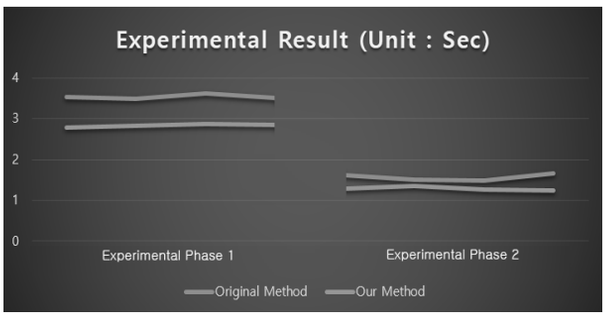


Fig. 11. Experimental Result Chart (Unit: Sec)

실험의 결과 기존의 모든 삼각형 쌍의 교차 테스트를 병렬로 수행하는 일반적인 방법(Original Method)은 Phase 1의 경우 평균 3.5384초, Phase 2의 경우 1.5643초의 수행 시간을 소모하였다. 그러나 본 연구를 통해 제안된 방법(Our Method)은 Phase 1의 경우 2.8328초, Phase 2의 경우 1.2863초의 수행 시간을 소모하였다. 이는 모델의 크기가 작은 Phase 2에서 평균 21.61%의 성능 개선을 가져왔으며, 모델의 크기가 큰 Phase 1에서 평균 24.9%의 성능 개선을 가져왔다. 이는 물체의 크기가 커질수록 더욱 효과적인 알고리즘임을 나타낸다.

5. 결 론

본 논문은 변형 물체 시뮬레이션의 성능 개선을 위한 GPU 기반 병렬 충돌 감지 알고리즘에 관한 연구 결과를 나타낸다. GPU의 성능이 상승함에 따라 더 높은 해상도의 변형 물체를 시뮬레이션 할 수 있으나, 이에 맞는 효과적인 병렬 알고리즘이 필요한 상황이다. 본 연구에서는 GPU 환경에 적합한 자료구조 설계 및 파이프라인 설계를 통해 1차적인 성능 개선을 수행하였으며, 바운딩 박스를 활용한 GPU 기반 컬링 알고리즘의 설계를 통해 충돌 검사 연산의 속도를 향상시킬 수 있었다. 더욱이 본 연구를 통해 변형 객체의 움직임을 GPU 기반 병렬처리 기법으로 처리할 때, 병렬성을 높이기 위한 SSBO의 구성과 각 컴퓨트 쉐이더의 컴퓨트 스페이스 구성방법에 대하여 제시하였다. 추후 본 연구의 결과를 바탕으로 다양한 충돌 검사 연산에 적용을 통한 성능 개선이 가능할 것으로 예상한다.

References

[1] Shreiner, Dave, et al., "OpenGL programming guide: The Official guide to learning OpenGL, version 4.3," 2013.

[2] B. Burger, S. Bettinghausen, M. Radle, and J. Hesser, "Real-time GPU-based ultrasound simulation using deformable mesh models," *IEEE Transactions on Medical Imaging*, Vol.32, No.3, pp.609-618, 2013.

[3] G. Echegaray, I. Herrera, I. Aguinaga, C. Buchart, and D. Borro, "A brain surgery simulator," *IEEE computer Graphics and Applications*, Vol.34, No.3, pp.12-18, 2014.

[4] Nak-Jun Sung, Min Hong, Seung-Hyun Lee, and Yoo-Joo Choi, "Simulation of Deformable Objects using GLSL 4.3," *KSII Transactions on Internet & Information Systems (TIIS)*, Vol.11, No.8, 2017.

[5] Imran Ghani, Min Sang Kim, Nak-Jun Sung, Min Hong, and Yoo-Joo Choi, "Real-Time Cloth Simulation using Unity Shader," in *Proceedings. of The 12th Asia Pacific International Conference on Information Science and Technology (APIC-IST2017)*, 2017.

[6] David Baraff and Andrew Witkin, "Large Steps in Cloth Simulation," in *Proceedings. of the 25th Annual Conference on Computer Graphics and Interactive Techniques,* pp.43-54, 1998.

[7] Hang Si, "TetGen, a Delaunay-based quality tetrahedral mesh generator," *ACM Transactions on Mathematical Software (TOMS)*, Vol.41, No.22, pp.11, 2015.

[8] D. Kim, J. P. Heo, J. Huh, J. Kim, and S. E. Yoon, "HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs," in *Computer Graphics Forum*, Vol.28, No.7, pp.1791-1800, 2009.

[9] M. Tang, D. Manocha, J. Lin, and R. Tong, "Collision-streams: fast gpu-based collision detection for deformable models" in *Symposium on Interactive 3D Graphics and Games ACM*, pp.63-70, 2011.

[10] J. Mezger, S. Kimmerle, and O. Etzmuss, "Hierarchical Techniques in collision detection for cloth animation," *Journal of WSCG*, Vol.11, No.2, pp.322-329, 2003.

[11] A. Borrmann, S. Schraufstetter, and E. Rank. "Implementing metric operators of a spatial query language for 3D building models: octree and B-Rep approaches," *Journal of Computing in Civil Engineering*, Vol.23, No.1, pp.34-46, 2009.

[12] X. Wang, M. Wang, and C. Li, "Research on Collision Detection Algorithm Based on AABB," *ICNC*, Vol.6, 2009.

[13] Y.-S. Xing, X. P. Liu, and S.-P. Xu, "Efficient collision detection based on AABB trees and sort algorithm," in *8th IEEE International Conference on Control and Automation (ICCA)*, 2010.

[14] Gino van den Bergen, "Efficient collision detection of complex deformable models using AABB trees," *Journal of Graphics Tools*, Vol.2, No.4, pp.1-13, 1997.

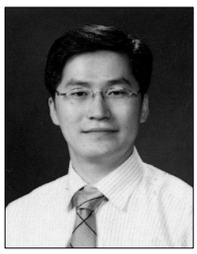
[15] J. H. Jeon, M. H. Choi, and M. Hong, "Enhanced FFD-AABB collision algorithm for deformable objects," *Journal of Information Processing Systems*, Vol.8, No.4, pp.713-720, 2012.

[16] X. Zhang, and Y. J. Kim, "Interactive collision detection for deformable models using streaming AABBs," *IEEE Transactions on Visualization and Computer Graphics*, Vol.13, No.2, pp.318-329, 2007.

[17] T. Moller, “A Fast Triangle-Triangle Intersection Test,” *Journal of graphics tools*, Vol.2, No.2, pp.25-30, 1997.
[18] M. Tang, S. E. Yoon, and D. Manocha, “Adjacency-based culling for continuous collision detection,” *Visual Computer*, Vol.24, No.7, pp.545-553, 2008.
[19] J. Barbič, and D. L. James, “Subspace self-collision culling,” *ACM Transactions on Graphics*, Vol.29, No.4, pp.81, 2010.

성 낙 준
http://orcid.org/0000-0003-3514-7152
e-mail : njsung@sch.ac.kr
2016년 순천향대학교 컴퓨터소프트웨어공학과(학사)
2017년~현 재 순천향대학교 컴퓨터학과 석사과정
관심분야 : Dynamic Simulation, Deformable Object Simulation, AR, VR, Bio Informatics

김 민 상
http://orcid.org/0000-0001-7320-1661
e-mail : ben399399@sch.ac.kr
2017년 순천향대학교 컴퓨터소프트웨어공학과(학사)
2017년~현 재 순천향대학교 컴퓨터학과 석사과정
관심분야 : Computer Graphics, Real Time Simulation



홍 민
http://orcid.org/0000-0001-9963-5521
e-mail : mhong@sch.ac.kr
1995년 순천향대학교 전산학과(학사)
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~현 재 순천향대학교 컴퓨터소프트웨어공학과 교수
관심분야 : Computer Graphics, Dynamic Simulation, Bio Informatics, Computer Vision

최 유 주
http://orcid.org/0000-0001-7520-097X
e-mail : yjchoi@smit.ac.kr
1989년 이화여자대학교 전자계산학과 (이학사)
1991년 이화여자대학교 전자계산학과 (이학석사)
2005년 이화여자대학교 컴퓨터공학과(공학박사)
1991년 (주)한국컴퓨터 기술연구소 주임연구원
1994년 (주)포스데이타 기술연구소 주임연구원
2005년 서울벤처정보대학원 컴퓨터응용기술학과 조교수
2010년~현 재 서울미디어대학원대학교 뉴미디어학부 부교수
2015년~현 재 서울미디어대학원대학교 실감미디어연구소 교수
관심분야 : Computer Graphics, Computer Vision, HCI, Augmented Reality