ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

워크 그룹 구성 변화에 따른 GPU 기반 천 시뮬레이션의 성능 분석☆

The Performance Analysis of GPU-based Cloth simulation according to the Change of Work Group Configuration

최 영 환[1] 홍 민[2] 이 승 현[3] 최 유 주[4*]
Young-Hwan Choi Min Hong Seung-Hyun Lee Yoo-Joo Choi

요 약

오늘날 3D 다이내믹 시뮬레이션은 많은 산업들과 밀접한 관계를 가지고 있다. 과거에는 자동차 충돌, 건축물 분야에서 주로 사용되었으나 최근에는 영화나 게임 분야에도 물리 시뮬레이션이 중요한 역할을 하고 있다. 일반적으로 3D 물체를 사실적으로 표현하기 위해서는 많은 수학적 연산이 필요하기 때문에 기존의 CPU 기반의 응용 프로그램들은 이러한 많은 연산량을 실시간으로 처리하는데 무리가 있다. 최근 그래픽 하드웨어의 발전과 아키텍쳐의 개선으로 GPU는 기존의 렌더링 연산뿐만 아니라 범용 목적의 연산 기능을 제공하고 있고 이러한 GPU를 활용하는 연구가 활발히 진행되고 있다. 본 논문에서는 GPU를 이용한 천 시뮬레이션 수행시 수행 성능을 최적화하기 위하여, GPU 셰이더의 실행 환경 변화에 따른 천 시뮬레이션 알고리즘의 수행 성능의 변화를 분석하였다. GPU를 이용한 천 시뮬레이션은 GLSL 4.3의 Compute shader를 사용하여 스프링 중심 알고리즘과 노드 중심 알고리즘을 PC기반으로 구현하였고, GLSL Compute shader의 다양한 워크 그룹 (Work Group) 크기와 차원 분배에 따른 연산 속도의 변화를 비교 분석하였다. 실험은 5,000 프레임까지 10회 반복 수행하여 FPS(Frame Per Second)의 평균을 구하여 진행하였다. 실행결과, 노드 중심의 알고리즘이 오히려 스프링 중심의 알고리즘 보다 빠른 수행속도를 보여 주었다.

☞ 주제어 : 천 시뮬레이션, GPU 병렬처리, 물리기반 시뮬레이션, GLSL 4.3

ABSTRACT

In these days, 3D dynamic simulation is closely related to many industries. In the past, physically-based 3D simulation was used mainly in the car crash or construction related fields, but it also plays an important role in movies or games today. Many mathematical computations are needed to represent the 3D object realistically, but it is difficult to process a large amount of calculations for simulation of application based on CPU in real-time. Recently, with the advanced graphic hardware and improved architecture, GPU can be utilized for the general purposes of computation function as well as graphic computation. Many approaches using GPU have been applied for various research fields. In this paper, we analyze the performance variation of two cloth simulation algorithms based on GPU according to the change of execution properties of GPU shaders in oder to optimize the performance of GPU-based cloth simulation. Cloth simulation is implemented by the spring centric algorithm and node centric algorithm with GPU parallel computing using compute shader of GLSL 4.3. We compare the performance of between these algorithms according to the change of the size and dimension of work group. The experiment is repeated to 10 times during 5,000 frames for each test and experimental results are provided by averaging of FPS. The experimental result shows that the node centric algorithm is executed in higher speed than the spring centric algorithm.

☞ keyword : Cloth simulation, GPU parallel processing, Physically-based simulation, GLSL 4.3

1. 서 론

오늘날 물리기반의 3D 시뮬레이션은 교육 및 산업적

1 Dept of Computer Science, Soonchunhyang University, Asan, 31538, Korea.
2 Dept of Computer Software Engineering, Soonchunhyang University, Asan, 31538, Korea.
3 School of Architectural Engineering, Hongik University, Sejong, 2639, Korea
4 Dept of Newmedia, Seoul Media Institute of Technology, Seoul, 07590, Korea.
* Corresponding author (yjchoi@smit.ac.kr)

[Received 21 December 2016, Reviewed 10 January 2017, Accepted 5 April 2017]
☆ 본 연구는 한국연구재단 이공학개인기초연구지원사업 기본연구지원사업(NRF-2015R1D1A1A01059304)에 의하여 수행되었음
☆ 본 연구는 순천향대학교 학술연구비 지원으로 수행하였음

측면에 큰 기여를 하고 있으며 영화, 게임, 건설, 의료 등의 많은 분야에서 중추적 역할을 하고 있다. 영화나 게임 분야에서는 특수 효과 및 3D 애니메이션이 중심이 되었으나 최근에는 객체를 현실적으로 표현하는데 중점을 두고 있다. 건설 분야에서는 물리적인 시뮬레이션을 통해 안정성을 확보한 건물 설계에 활용하고 있으며 의료 분야에서는 분자 수준의 분석에서부터 골격 시뮬레이션 분야까지 다양한 분야에 사용되고 있다.

최근에 출시되고 있는 GPU는 매우 강력하고 유연한 프로세서로 진화하고 있다. 최신 그래픽 카드의 아키텍처는 부동 소수점의 벡터 연산을 지원하고 프로그래밍 가능한 정점 및 화소 처리 장치와 함께 큰 메모리 대역폭과 빠른 연산 속도를 제공한다. 구조적으로 GPU는 벡터 연산의 병렬처리에 최적화되어 있으며 GLSL(OpenGL Shading Language)를 통해 GPU 파이프라인에 접근 할 수 있다. 따라서 GPU를 단순한 그래픽 렌더링 목적이 아닌 일반적인 범용 목적으로 사용하는 기술이 주목받고 있으며, 이 기술은 많은 작업을 병렬로 처리하여 연산 속도를 향상 시킬 수 있다[1].

본 논문에서는 GPU 기반으로 천 시뮬레이션의 성능을 최적화하기 위해, OpenGL 4.3 버전부터 제공하는 GPU 병렬처리 기법을 활용하여 물리기반의 천 시뮬레이션 시스템을 스프링 중심과 노드 중심의 2가지 알고리즘으로 구현하여 처리 속도의 결과를 비교하였고, 워크 그룹(Work Group)의 차원, 크기에 따른 연산 속도의 변화를 비교 분석하였다.

2. 관련연구

2.1 Mass-spring 모델

3D 물체를 현실적으로 표현하기 위해서는 적절한 시뮬레이션 방법이 필요하다. 시뮬레이션에서 현실 세계의 물체들은 크게 강체와 변형물체로 구분할 수 있다. 강체는 외부의 힘을 가해도 모양이 거의 변하지 않기 때문에 표현이 간단하며 시뮬레이션 연산량 또한 비교적 적다. 반면에 변형물체는 외부의 힘에 의해 물체의 모양이 매 순간마다 변할 수 있으므로 이를 정확하게 표현하기 위해서는 많은 연산량을 필요로 한다. 일반적으로 변형물체의 시뮬레이션은 유한요소법(Finite Element Method)과 Mass-spring 모델을 사용한다. 일반적으로 실시간 시뮬레이션은 연산량이 많이 소모되는 유한요소법에 비해서 연산량이 적은 Mass-spring 모델이 널리 사용되고 있다[2-7].

Mass-spring 모델은 기본적으로 Structal, Shear, Bend의 3가지 구조로 구성된다[8, 9]. Structual 구조는 x, y축으로 인접한 노드를 스프링으로 연결하여 전체적인 천의 모양을 유지하고 Shear 구조는 대각선에 위치한 노드를 연결하여 대각 방향으로 힘을 주었을 때, 천의 모양을 유지한다. Bend 구조는 x, y축으로 2단계로 인접한 노드들을 연결하여 접거나 구부렸을 때 천의 모양을 유지한다. 또한 Mass-spring 모델은 후크의 법칙을 적용하여 다음 단계의 노드의 가속도, 속도, 위치를 계산한다.

2.2 Compute Shader

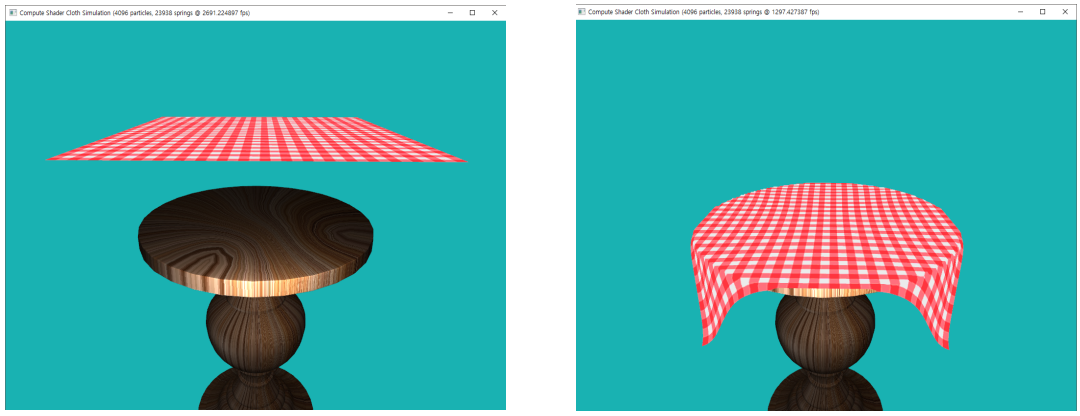
OpenGL은 크로노스 그룹에서 제정한 컴퓨터 그래픽스 규격으로 2D, 3D 그래픽 오픈소스 라이브러리이다. OpenGL 4.3은 2012년 8월에 배포되었고 기존에 없던 Compute shader를 추가하였다. Compute shader는 기존의 그래픽 파이프라인과 달리 독립된 쉐이더로서 임의의 정보를 계산하는 용도로 사용된다[10, 11].

Compute shader는 GLSL 4.3 이상의 버전의 프로그램으로 접근 할 수 있으며 연산 속도를 향상시키기 위하여 GPU에서 병렬처리를 수행할 수 있다. 작업량을 사용자가 지정한 워크 그룹의 크기만큼 X, Y, Z의 3차원 작업공간으로 나누어 할당이 가능하다. 또한 Compute shader는 할당된 각 워크 그룹에서 동작한다. 본 논문에서는 각 노드에 연결된 스프링들의 힘과 노드의 위치를 Compute shader를 통해서 계산을 하였다.

2.3 워크 그룹

워크 그룹은 Compute shader가 동작하는 작업 단위를 말한다. 3차원으로 이루어진 워크 그룹 전체를 전역 워크 그룹 (Global Work Group)이라고 하고 각각의 원소를 Local 워크 그룹이라고 한다. 워크 그룹은 glDispatch Compute()가 호출될 때 하나의 전역 워크 그룹이 Compute shader에 전달되며 전달된 전역 워크 그룹은 지역 워크 그룹(Local Workgroup)으로 나누어진다.

워크 그룹의 크기는 X, Y는 최대 1,024, Z는 최대 64의 크기를 가질 수 있으며 하나의 워크 그룹은 최소 1,024개의 스레드를 가질 수 있으며, GPU에 따라서 최대 가질 수 있는 스레드의 수가 달라진다. GPU의 최대 스레드의 수는 GL_MAX_COMPUTE_WORK_GROUP_SIZE를 통해서 알 수 있다. 각 워크 그룹은 서로 공유 가능한 메모리와 워크 그룹의 크기만큼의 스레드를 가지며 완전히 독립적



(그림 1) 천 시뮬레이션 결과
(Figure 1) The result of cloth simulation

인 구조를 가진다.

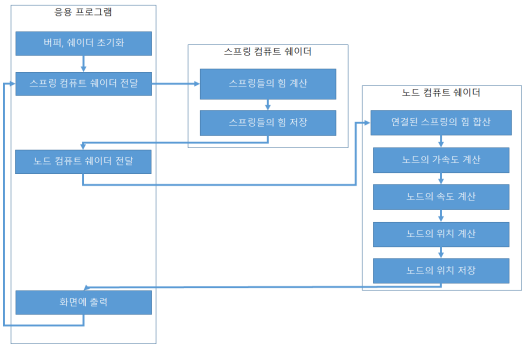
워크 그룹은 임의의 순서로 동작한다. 예를 들어 워크 그룹의 크기를 (512, 1, 1)로 구성하였을 때 워크 그룹 (1, 1, 1), (2, 1, 1), (3, 1, 1)의 순으로 동작하는 것이 아니라 비선형 방식으로 동작한다. 이러한 이유로 워크 그룹을 분배할 때, 비선형 구조로 작업량을 분배하는 것을 권장한다.

3. 천 시뮬레이션 구현

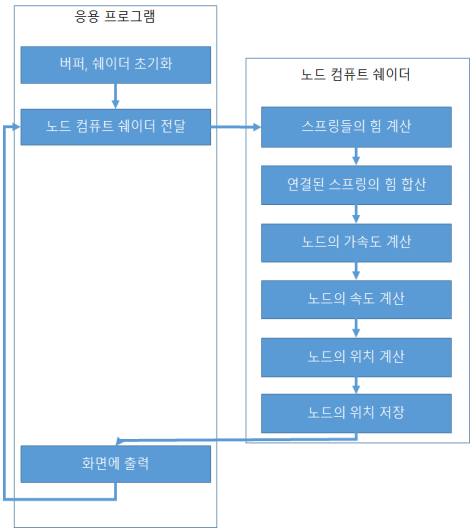
본 논문에서는 GLSL 4.3의 Compute shader를 사용하여 천 시뮬레이션을 구현하였다. 본 연구에서는 테이블의 일정 높이 위에 천을 위치시키고 테이블 위로 자유낙하 시켜 떨어트리는 시뮬레이션을 구현하였다. 그림 1은 천 시뮬레이션의 초기 화면과 테이블에 천이 떨어지는 시뮬레이션 화면이다.

3.1 스프링 중심 알고리즘

스프링 중심 알고리즘은 스프링의 힘을 계산하는 부분과 노드의 위치를 계산하는 부분으로 나눈다. 먼저 응용 프로그램에서 버퍼와 쉐이더를 초기화 시키고 Spring Compute shader에 작업을 워크 그룹의 단위로 전달한다. Spring Compute shader는 전역 워크 그룹을 지역 워크 그룹로 나누고 각각의 워크 그룹 내의 스레드에 작업을 할당한다. 각각의 스레드는 할당된 스프링의 힘을 계산하고 SSBO(Shader Storage Buffer Object)에 저장하면서 스레드를 종료한다. Spring Compute shader의 모든 스레드가 종료되면 응용 프로그램에서는 Node Compute shader에 워크 그룹을 할당한다. Node Compute shader에서는 노드에 연결된 스프링들의 힘을 합산하여 노드에 가해지는 힘을 구한다[12, 13]. 구해진 힘으로 노드의 가속도를 계산하고 계산된 가속도를 적용하여 노드의 속도를 계산한다. 마지막으로 계산된 속도를 이용해 노드의 위치를 계산 할 수 있다. 계산된 노드의 위치는 SSBO에 저장된 후에 스레드가 종료된다. 응용 프로그램에서는 SSBO에 저장된 노드의 위치를 화면에 렌더링한다. 그림 2는 스프링 중심의 천 시뮬레이션에 대한 프로세스 구조를 나타난 것이다. 스프링 중심 알고리즘은 노드 중심 알고리즘 보다 복잡한 구조를 가지지만 유지, 보수가 쉽고 연산량이 적다는 장점을 가진다.



(그림 2) 스프링 중심의 천 시뮬레이션 구조
(Figure 2) The spring centric structure of cloth simulation



(그림 3) 노드 중심의 천 시뮬레이션 구조
(Figure 3) The node centric structure of cloth simulation

3.2 노드 중심 알고리즘

노드 중심 알고리즘은 Compute shader에 스프링의 힘과 노드의 위치를 계산하는 부분을 합친 구조를 가진다. 먼저 응용 프로그램에서 버퍼와 쉐이더를 초기화 시키고 Node Compute shader에 작업을 워크 그룹의 단위로 전달하여 각각의 워크 그룹의 스레드에서는 노드에 연결된 스프링의 힘들을 계산하고 합하여 노드에 가해지는 힘을 구한다. 구해진 힘으로 노드의 가속도를 계산하고 계산된 가속도로 노드의 속도를 계산한다. 계산된 속도로 노드의 위치를 계산하고 SSBO에 저장하고 스레드가 종료된다. 응용 프로그램에서는 SSBO에 저장된 노드의 위치를 화면에 렌더링한다. 그림 3은 노드 중심의 천 시뮬레이션의 구조를 나타낸 것이다. 노드 중심 알고리즘은 스프링 중심 알고리즘보다 구조가 간단하지만 스프링의 힘을 계산하기 위해 노드를 중심으로 연결된 스프링에 대해 한 번씩 추가로 연산을 해야 하는 단점을 가진다.

(표 1) 실험 환경
(Table 1) Test environment

CPU	Intel Core i7-4790K
GPU	Nvidia Geforce GTX 970 4GB
RAM	DDR3 16GB
OS	Windows 10 Pro KN
GPU driver	369.09

4. 실험 환경 및 결과

본 논문에서 구현된 GPU를 이용한 병렬처리 천 시뮬레이션 시스템의 실험 환경은 다음 표 1과 같다. 실험 방법은 0번 프레임부터 5,000번 프레임까지 10번 반복하여 FPS(Frame Per Second)의 평균을 구하였다. 또한 GPU의 최대 연산속도를 비교하기 위하여 그래픽 카드에 설정된 수직 동기화를 해제하였다.

표 2와 표 3은 스프링 중심 알고리즘과 노드 중심 알고리즘을 적용한 실험에서 노드의 수와 워크 그룹의 차원과 크기별 시뮬레이션 결과에 대한 FPS 평균을 나타낸 것이다. 같은 노드 수의 결과 중에서 가장 빠른 FPS를 나타낸 결과는 어두운 색 배경으로 지정하였다. 스프링 중심 알고리즘에서는 워크 그룹의 크기를 32이고 1차원으로 할당했을 때 가장 빠른 연산 속도를 보였고, 워크 그룹의 크기가 동일할 때에는 1차원으로 할당하였을 때 빠른 결과를 보였다. 이러한 결과를 나타낸 이유는 워크 그룹의 크기가 커지면서 스레드가 많아지게 되므로 스레드 간의 동기화를 위한 시간이 길어지기 때문으로 예측된다. 반대로 노드 중심 알고리즘에서는 워크 그룹의 크기를 256으로 3차원 (16, 8, 2)으로 할당했을 때 대부분 빠른 연산 속도를 보였다. 두 알고리즘의 결과를 비교하였을 때 워크 그룹의 크기가 32이고 (2, 2, 8)의 크기로 할당 했을 때, 노드의 개수가 256×256, 512×512의 경우를 제외하고 모든 경우에서 노드 중심 알고리즘이 더 빠른 결과를 나타내었다. 노드 중심 알고리즘의 연산량이 더 많음에도 더 빠른 결과를 보인 이유는 스프링 중심 알고리즘에서는 Compute shader 간의 동기화 시간이 더 길기 때문으로 분석된다.

(표 2) 스프링 중심 알고리즘의 천 시뮬레이션 결과
(Table 2) The result of cloth simulation based on spring centric algorithm

천 시뮬레이션	x, y, z축의 워크그룹 크기			노드 수 별 FPS 평균(노드 중심)				
워크그룹의 전체 크기	x축	y축	z축	노드수	256*256 (65,536)	512*512 (262,144)	1,024*1,024 (1,048,576)	2,048*1,024 (2,097,152)
				스프링수	390,658	1,567,746	6,281,218	12,567,554
32	32	1	1	FPS	846.77	273.65	74.11	37.74
	4	8	1		698.83	203.48	48.34	23.22
	2	2	8		576.90	163.53	38.26	18.27
64	64	1	1		744.31	232.73	62.4	31.87
	2	32	1		470.81	122.48	30.58	15.45
	4	16	1		587.36	163.69	40.56	20.00
	2	2	16		501.22	126.75	30.87	15.41
	4	4	4		591.49	173.24	44.23	22.17
128	128	1	1		746.00	233.09	62.64	32.07
	2	64	1		463.74	120.84	30.17	15.22
	8	16	1		656.44	188.78	47.43	23.46
	2	2	32		479.20	120.01	30.01	15.10
	4	4	8		592.05	169.57	41.84	20.49
256	256	1	1		748.42	235.04	63.26	32.26
	2	128	1		496.11	124.53	30.49	15.39
	16	16	1		707.80	209.76	53.34	26.29
	2	2	64		447.65	117.01	29.46	14.93
	16	8	2		730.60	224.07	59.54	30.32
512	512	1	1		741.27	229.11	61.35	31.33
	2	256	1		493.12	133.83	30.83	15.34
	16	32	1		682.74	194.42	50.52	25.5
	4	2	64		536.70	146.03	37.23	18.57
	16	16	2		718.91	218.57	56.98	28.58
1,024	1,024	1	1		735.88	225.87	60.22	30.77
	2	512	1		495.55	130.68	32.92	15.90
	32	32	1		706.67	197.74	50.93	25.79
	4	4	64		544.28	147.66	36.11	18.26
	32	16	2		726.22	221.47	57.46	29.37
				최댓값	846.77	273.65	74.11	37.74
				최솟값	447.65	117.01	29.46	14.93
				평균값	623.82	180.27	46.15	23.18

(표 3) 노드 중심 알고리즘의 천 시뮬레이션 결과
(Table 3) The result of cloth simulation based on node centric algorithm

천 시뮬레이션	x, y, z축의 워크그룹 크기			노드 수 별 FPS 평균(노드 중심)				
워크그룹의 전체 크기	x축	y축	z축	노드수	256*256 (65,536)	512*512 (262,144)	1,024*1,024 (1,048,576)	2,048*1,024 (2,097,152)
				스프링수	390,658	1,567,746	6,281,218	12,567,554
32	32	1	1	FPS	957.32	310.18	84.55	42.09
	4	8	1		786.86	245.26	65.71	33.39
	2	2	8		526.04	150.84	39.42	19.92

(표 3) 노드 중심 알고리즘의 천 시뮬레이션 결과(계속)
(Table 3) The result of cloth simulation based on node centric algorithm

64	64	1	1	FPS	932.62	299.83	81.10	41.17
	2	32	1		523.69	150.88	39.41	20.00
	4	16	1		789.07	245.46	65.55	33.36
	2	2	16		524.01	150.69	39.38	19.87
	4	4	4		789.27	244.90	65.60	33.36
128	128	1	1		929.86	300.11	81.16	41.28
	2	64	1		524.35	150.81	39.41	19.87
	8	16	1		1,048.89	356.22	98.09	49.84
	2	2	32		523.74	150.84	39.40	19.89
	4	4	8		788.11	245.29	65.62	32.14
256	256	1	1		934.76	300.68	81.33	41.30
	2	128	1		523.48	150.92	39.39	19.82
	16	16	1		1,160.99	428.65	118.10	57.93
	2	2	64		524.16	150.73	39.42	19.85
	16	8	2		1,179.65	423.81	119.72	58.13
512	512	1	1		943.83	302.56	81.47	41.42
	2	256	1		524.93	150.97	39.40	19.82
	16	32	1		1,160.76	425.74	118.52	57.86
	4	2	64		790.97	244.97	65.68	33.35
	16	16	2		1173.24	424.57	118.99	57.98
1,024	1,024	1	1		957.73	312.62	85.35	43.13
	2	512	1		524.58	151.04	39.36	19.84
	32	32	1		949.43	310.20	84.47	42.28
	4	4	64		788.29	245.19	65.51	33.32
	32	16	2		951.75	309.72	84.47	35.96
				최댓값	1,179.65	425.74	119.72	58.13
				최솟값	523.48	150.69	39.36	19.82
				평균값	808.02	252.98	70.91	35.29

5. 결 론

본 논문에서는 OpenGL 4.3부터 지원하는 GPU 병렬처리 기법을 사용하여 천 시뮬레이션을 구현하였다. 천 시뮬레이션은 스프링 중심 알고리즘과 노드 중심 알고리즘으로 구현하였고 두 알고리즘의 연산 속도를 비교하였다. 또한 각 워크 그룹의 크기와 1, 2, 3차원으로 할당했을 때의 연산 속도도 비교하였다.

스프링 중심 알고리즘에서는 워크 그룹의 크기가 작고 1차원으로 할당 할수록 더 빨랐고 노드 중심 알고리즘에서는 워크 그룹의 크기가 256이고 3차원으로 할당하였을 때 빠른 결과를 보였다. 또한 노드 중심 알고리즘이 스프링 중심 알고리즘보다 더 빠른 결과를 보였다. 따라서 같은 작업량이 있을 때, 알고리즘의 특성에 맞게 워크 그룹의 크기와 차원을 적절하게 분배하여 최적화한다면 더 좋은 결과를 볼 수 있을 것으로 기대한다.

참 고 문 헌(Reference)

[1] Wikipedia, "General-purpose computing on graphics processing units," https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units.

[2] Baraff David, Andrew Witkin, "Large Steps in Cloth Simulation," COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998, pp. 19-24. http://dx.doi.org/10.1145/280814.280821

[3] Gibson, Sarah F, "3D chainmail: a fast algorithm for deforming volumetric objects," Proceedings of the

1997 symposium on Interactive 3D graphics, 1997, pp. 149-154.
http://dx.doi.org/10.1145/253284.253324

[4] Vassilev, Ivanov Tzvetomir, Bernhard Spanlang, “A mass-spring model for real time deformable solids,” In Proceedings of the East-West Vision, 2002, pp. 149-154.
http://dx.doi.org/10.1109/ICMT.2011.6001862

[5] Kim Junho, Seung-Hyun Yoon, Yunjin Lee, ″Trivariate B-spline Approximation of Spherical Solid Objects,″ JIPS Vol. 10, no. 1, 2014, pp. 23-35.
http://dx.doi.org/10.3745/JIPS.2014.10.1.023

[6] Y. Chen, Qing-Hong Zhu, A. Kaufman, S. Muraki, “Physically-based animation of volumetric objects,” In Computer Animation, 1998, pp. 154-160.
http://dx.doi.org/10.1109/CA.1998.681920

[7] Kurzion, Yair, Roni Yagel, ″Space deformation using ray deflectors,″ Rendering Techniques’ 95, Springer Vienna, 1995, pp. 21-30.
http://dx.doi.org/10.1007/978-3-7091-9430-0_3

[8] Bianchi, Gérald, Matthias Harders, and Gábor Székely, ″Mesh topology identification for mass-spring models,″ International Conference on Medical Image Computing and Computer-Assisted Intervention, 2003.
http://dx.doi.org/10.1007/978-3-540-39899-8_7

[9] Xavier provot, “Deformation Constraints in a Mass -Spring Model to Describe Rigid Cloth Behavior”, Graphics Interface, pp.147-155, 1995.
http://dx.doi.org/10.1029/JB094iB03p03065

[10] OpenGL WIKI, “Compute shader,”
https://www.opengl.org/wiki/Compute_Shader

[11] Mark Segal and Kurt Akeley, “The OpenGL Graphics System:A Specification (Version 4.3 (Core Profile)),” The Khronos Group Inc, 2012.

[12] Provot, Xavier, “Deformation constraints in a mass-spring model to describe rigid cloth behaviour,” Graphics interface, Canadian Information Processing Society, 1995, pp. 147-147.
http://dx.doi.org/10.20380/GI1995.17

[13] Ro Man Hwang, Soo Kyun Kim, Syungog An, Dong-Won Park, ″The architectural pattern of a highly extensible system for the asynchronous processing of a large amount of data,″ Journal of Information Processing Systems 9.4, 2013, pp. 567-574.
http://dx.doi.org/10.3745/JIPS.2013.9.4.567

◐ 저 자 소 개 ◑

최 영 환(Hwan-Hwan Choi)
2015년 순천향대학교 컴퓨터소프트웨어공학과 (공학사)
2015~현재 순천향대학교 대학원 컴퓨터학과 석사 과정
관심분야 : 다이나믹시뮬레이션, 영상처리
E-mail : compust@sch.ac.kr

홍 민(Min Hong)
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, 다이나믹 시뮬레이션, 바이오 인포매틱스, 영상처리
E-mail : mhong@sch.ac.kr

이 승 현(Seung-Hyun Lee)
1996년 인하대학교 건축공학과(공학사)
1999년 University of Colorado at Boulder(공학석사)
2003년 University of Florida at Gainesville(공학박사)
2004년~2008년 건설기술연구원 선임연구원
2008년~현재 홍익대학교 건축공학과 교수
관심분야 : 프로세스 시뮬레이션, 생산성, 최적화, 영상처리, 자동 데이터 수집
E-mail : slee413@hongik.ac.kr

최 유 주(Yoo-Joo Choi)
1989년 이화여자대학교 전자계산학과(이학사)
1991년 이화여자대학교 일반대학원 전자계산학과(이학석사)
2005년 이화여자대학교 과학기술대학원 컴퓨터공학과(공학박사)
1991년~1993년 한국컴퓨터주식회사 기술연구소 주임연구원
1994년~1999년 포스데이타주식회사 기술연구소 주임연구원
2005년~2010년 서울벤처대학원대학교 컴퓨터응용기술학과 조교수
2010년~현재 서울미디어대학원대학교 뉴미디어콘텐츠학과 교수
관심분야 : 컴퓨터 그래픽스, 모바일 증강현실, 영상인식, 영상보정
E-mail : yjchoi@smit.ac.kr