CUDA 를 이용한 가상 객체들간의 병렬 충돌 검사 알고리즘

Parallel Intersection Detection Algorithm using CUDA

이연희, Yeonhee Lee*, 김영준, Young J. Kim**

요약 CUDA 는 GPGPU 프로그래밍을 위해 nVIDIA 사에서 개발한 병렬 처리 프로그래밍 개발 환경이다. 본 논문에서는 가상 객체들 간의 삼각형 충돌 검사 부분을 CUDA 를 이용해 병렬적으로 구현하였다. 삼각형 충돌 검사는 실시간 충돌 검사 시 주요 병목현상을 일으키는 부분이다. 하지만 CPU 와 GPU 간의 데이터 전송 지연 문제 때문에 기존의 오브젝트 스페이스상의 GPU 기반의 충돌 검사 방법으로는 이 병목현상을 해결하기 어려웠다. 그러나 데이터 전송 지연 문제를 크게 완화시킨 CUDA 를 이용해 데이터 전송에 소모되는 비용을 줄이고 또한 삼각형 충돌 검사를 병렬적으로 수행함으로써 가상 객체를 형성하는 삼각형 집합들의 충돌검사 알고리즘의 성능을 크게 향상시킬 수 있었다.

Abstract In this paper, we present how we implement the low-level, triangle intersection routine in a massively parallel fashion using nVIDIA's new GPGPU language, CUDA. Triangle intersection often becomes a computational bottleneck in the collision detection problem. Due to the relatively low bandwidth between CPU and GPU, it has been challenging to implement efficient, object-space collision detection between triangle sets. However, thanks to the improved data transmission rates in CUDA architecture, in this paper, we improved the performance of triangle intersection substantially better than the optimized CPU counterpart.

핵심어: GPGPU, CUDA, Collision Detection, Triangle Intersection Test

본 논문은 한국학술진흥재단의 신진교수 연구과제(KRF-2007-D00400)에 의하여 지원되었음.

*이연희: 이화여자대학교 컴퓨터정보통신공학과 석사과정 e-mail: yeony102@ewhain.net

**김영준: 이화여자대학교 컴퓨터정보통신공학과 교수 e-mail: kimy@ewha.ac.kr

1. 서론

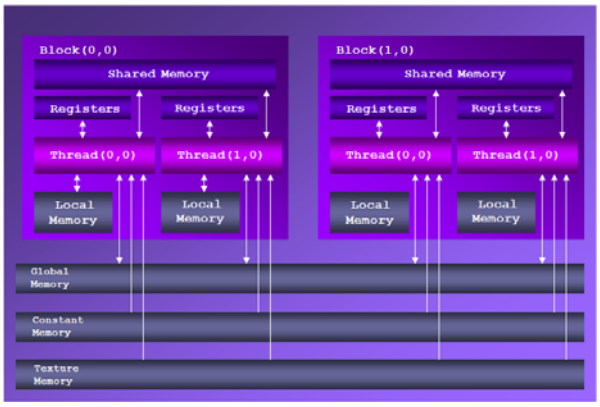
실시간 컴퓨터 게임 및 인터랙티브 그래픽스 기술에 대한 수요가 증가하면서 그래피컬한 영상 처리를 빠르게 할 수 있는 그래픽스 하드웨어(GPU)에 대한 요구가 최근 급격하게 증대하였다. 그 결과 오늘날의 GPU 는 CPU 에 비해 부동 소수점 처리를 비롯한 여러 부분의 연산 능력에서 훨씬 앞선 성능을 발휘하게 되었다. 따라서 병렬 처리로 연산 횟수를 크게 줄일 수 있는 연산 또는 빠른 데이터 처리를 요구하는 실시간 연산 처리에 GPU 를 이용한다면 효율적인 결과를 얻을 수 있다.

GPGPU(General Purpose computing on GPU)란 Graphics Hardware 혹은 GPU 를 그래픽 연산 처리가 아닌 범용 연산의 용도로 활용하는 것을 의미한다. 예를

들어 선형대수 계산, 이미지 프로세싱, 물리 기반 모델링, 데이터 베이스 연산 등 다양한 분야에 최근 들어 GPU 를 응용하는 연구들이 활발히 진행되고 있다. 이런 추세에 맞추어 최근 GPU 개발사들이 GPGPU 애플리케이션 개발을 효율적으로 하기 위한 각종 도구 및 아키텍처를 개발하고 있는데 그 중 대표적인 것이 nVIDIA 사의 CUDA 이다.

CUDA(Compute Unified Device Architecture) 는 nVIDIA GPU 를 위한 통합된 병렬 데이터 연산을 제공하는 소프트웨어 개발 도구이다. CUDA 는 병렬 연산을 위한 프로그래밍 단위로 커널(kernel), 그리드(grid), 블록(block) 및 스레드(thread)를 제공한다. 이 중 커널은 병렬 처리 해야 할 연산을 담은 함수로서, GPU 가 호출하는 기본적인 프로그램을 담고 있는 단위이다. 블록은 서로 의사소통이 가능한 스레드들의 집합이며, 같은 커널을 실행시키는 블록들이 모여서 그리드를 이룬다.

특히 CUDA 의 큰 특징은 공유 메모리(shared memory)의 사용에 있다. 이 공유 메모리는 온칩(On-chip) 메모리로서 로컬 메모리나 글로벌 메모리 같은 장치 메모리(device memory)에 접근하는 것보다 훨씬 빨리 접근할 수 있다. 공유 메모리는 동일 블록내의 스레드들이 공유하며, CUDA 는 이 공유 메모리를 이용해 데이터 접근 시간을 단축시킴으로써 기존의 CPU 보다 연산을 더 빨리 처리할 수 있다 (그림 1).



〈그림 1〉

컴퓨터 그래픽스, 가상현실, 햅틱스, 로보틱스 등의 애플리케이션에서 충돌 검사는 가상의 객체와 그 객체가 일으키는 물리적 사실감을 유지하게 하는 중요한 기술이다. 이러한 충돌검사에 일반적으로 쓰이는 방법은 AABB 나 OBB 등과 같은 BVH(Bounding Volume Hierarchy)를 이용한 알고리즘이다. 이런 알고리즘들은 객체를 구성하는 기본 도형인 삼각형들을 계층 구조로 구조화한 뒤, 계층적인 충돌 검사로 충돌 가능성이 있는 최소 삼각형 집합만을 추려내는 과정을 거치기 때문에 불필요한 삼각형 레벨에서의 연산을 줄일 수 있다. 하지만 변형 모델(deformable model) 혹은 파쇄(fracturing) 효과를 동반하는 모델의 충돌 검사를 할 경우엔 위의 BVH 기반의 컬링의 과정을 거치더라도 false positive 가 많이 생성되어 충돌 검사를 해야 할 삼각형 집합이 많아 진다. 이런 경우엔 삼각형 충돌 검사를 수행하는 부분이 전체 시스템의 연산 시간을 지연시킨다 [1].

이러한 삼각형 쌍들 간의 충돌 검사의 경우 각 쌍들의 충돌 검사는 병렬적으로 수행이 가능하며 이는 SIMD(Single Instruction Multiple Data) 방식의 병렬 처리에 적합하다. 따라서 병렬 연산으로 많은 삼각형 쌍에 대한 충돌 검사를 동시에 처리한다면 가상 객체 간 충돌 검사 시스템의 성능을 크게 향상시킬 수 있다. 본 논문은 앞서 소개한 CUDA 를 이용하여 삼각형 집합의 충돌 검사를 병렬적으로 구현하였다.

2. 관련 연구

[Tomas Möller 1997]은 CPU 기반의 효율적인 삼각형 충돌검사 알고리즘[2]을 제시한다. 이 알고리즘은 현재 알려진 삼각형 간 충돌 검사 알고리즘 중 가장 효율적이기 때문에 많은 충돌 검사 알고리즘들이 삼각형 간 충돌 검사 단계에서 이를 이용한다. 하지만 검사해야 할 삼각형의 개수가 아주 많은 경우에는 이 알고리즘 역시 모든 삼각형 쌍에 대하여 연산하는데 상당 시간이 소요된다.

[Choi et al 2004]은 가상 모델 간 충돌 검사 시 삼각형 간 충돌 검사 단계에서 GPGPU 를 활용한다. 이를 CPU 로 구현했을 때와는 달리 삼각형 쌍의 수가 많아져도 연산 시간이 크게 지연되지 않지만 구현에 이용한 GPU 의 한계로 인해 1000 개 이상의 삼각형을 갖는 복잡도가 높은 모델에는 적용하기 어렵다는 문제점이 있다. 또한 이 구현에 이용한 GPU 는 GPGPU 를 위한 기능을 별도로 갖지 않기 때문에 병렬 연산 구현을 위해서는 따로 쉐이딩 언어를 익혀야 하는데, 쉐이딩 언어는 랜더링을 목적으로 만들어진 언어이므로 일반적인 계산을 하기에는 직관적이지 않으며, GPU 를 GPGPU 계산에 최적화 하기 힘들다는 문제를 갖는다.

3. CPU 기반의 삼각형 충돌검사 알고리즘

본 논문은 Tomas Möller 가 제안한 CPU 기반의 효율적 삼각형 충돌검사 알고리즘[2]을 사용하여 삼각형 충돌 검사 함수를 구현하였다.

각각 정점 v_0, v_1, v_2 의 집합과 정점 u_0, u_1, u_2 의 집합을 꼭지점으로 하는 삼각형 T_1 과 T_2 가 있다고 할 때, 이 두 삼각형에 대해 충돌 검사를 한다고 가정하면 우선 삼각형 T_2 를 포함하는 평면의 방정식 P_2 을 다음의 식으로 구한다.

$$P_2 : \ N_2 \cdot x + d_2 \qquad (1)$$

$$N_w = (u_1 - u_0) \times (u_2 - u_0) \qquad (2)$$

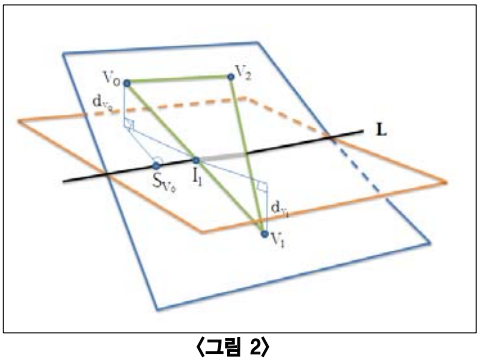
$$d_2 = -N_2 \cdot u_0 \qquad (3)$$

그리고 삼각형 T_1 의 꼭지점들로부터 평면 P_2 까지의 거리

$$d_{Vi} = N_2 \cdot v_i + d_2, \quad i = 0,1,2 \qquad (4)$$

를 구할 수 있다. 만약 $d_{Vi} \neq 0,\ i = 0,1,2$ 이고 모두 같은 부호를 갖는다면 삼각형 T_1 은 평면 P_2 의 half space 에 포함된다는 것을 의미하므로 충돌 검사 결과를 거짓으로 보고하고 종료한다. 같은 과정을 정점 u_0 , u_1 , u_2 와 삼각형 T_1 에 대해서도 반복한 후, 이 검사를 통과한 삼각형 쌍에 대해서는 다음과 같은 검사를 실시한다.

두 평면 P_1 과 P_2 가 충돌하는 직선을 L 이라고 할 때 직선 L 과 각각의 삼각형 T_1 , T_2 가 충돌하는 선분을 구한다. 이 두 선분이 겹쳐지면 삼각형 T_1 과 T_2 는 충돌하는 것이고, 겹치지 않는다면 충돌하지 않는 것이다.



〈그림 2〉

〈그림 2〉에서 삼각형 T_1 과 직선 L 이 교차하는 선분의 한 쪽 끝점을 구하는 식은 다음과 같다

$$I_1 = S_{V_0} + \left(S_{V_1} - S_{V_0}\right)\frac{d_{V_0}}{d_{V_0} - d_{V_1}} \qquad (5)$$

$$S_{V_i} = D \cdot \left(v_i - A\right) \qquad (6)$$

$$D = N_1 \times N_2 \qquad (7)$$

식 (6)에서 A 는 D 위의 어떤 점이다. 위와 비슷한 방법으로 다른 끝점도 구하고, 삼각형 T_2 와 직선 L 과의 교차점들도 모두 구하면 두 선분의 교차 여부를 알 수 있다.

4. CUDA 를 이용한 병렬 충돌 검사

CUDA 를 이용해 삼각형 충돌 검사를 구현하기 위해서는 위의 효율적 삼각형 충돌 검사 알고리즘을 병렬 처리화 해야 한다. 이번 절에서는 병렬 연산을 위한 자료 구조와 알고리즘에 대해 설명한다.

4.1 병렬 자료 구조

CUDA 프로그래밍을 위해서는 충돌연산에 사용할 스레드의 개수와 스레드 블록의 크기를 정해 주어야 한다. 충돌 검사를 할 두 개의 객체 O_1 과 O_2 를 이루는 삼각형의 개수를 각각 n_1 과 n_2 라고 했을 때, 이 알고리즘을 수행할 그리드 내의 총 스레드 개수는 $(n_1 \times n_2)$ 개가 된다. 각 스레드는 각각 한 쌍의 삼각형에 대해 충돌 검사를 수행한다. 임의의 스레드가 맡는 한 쌍의 삼각형 조합은 그리드 내의 그 스레드 위치에 따라 결정된다. 그리드는 그 구성이 추상적으로 $(n_1 \times n_2)$ 개의 2 차원 행렬 모양으로 스레드가 세로로 n_1 개, 가로로 n_2 개 정렬 되어 있는 것과 같다. 예를 들어 세로로 i 번째, 가로로 j 번째에 위치한 스레드가 충돌 검사를 할 삼각형 쌍은 객체 O_1 의 i 번째 삼각형과 객체 O_2 의 j 번째 삼각형의 조합이 된다.

스레드 블록의 크기는 한 블록 내의 스레드 개수를 의미하는데, 이에 따라 한 그리드 내의 블록 개수도 정해진다. 스레드 블록의 크기는 2 차원 또는 3 차원으로 정의할 수 있는데, 이 알고리즘을 위한 스레드 블록의 크기는 2 차원으로 정의하였다. 스레드 블록의 세로 크기 l_{hig} 와 가로 크기 l_{wid} 는 삼각형 개수 n_1 과 n_2 에 따라 달리 정의되며, 따라서 그리드 내의 블록 개수는

$$\left(\frac{n_1}{l_{hig}}\right) \times \left(\frac{n_2}{b_{wid}}\right)$$

가 된다.

4.2 병렬 충돌 검사 알고리즘

병렬처리의 호스트에 해당하는 CPU 는 먼저 각 객체들을 이루는 삼각형들에 대한 정점의 좌표를 읽어 들여서 호스트 변수에 저장한다. 그리고 커널 함수 호출 전, GPU 상에 그 호스트 변수들과 같은 크기의 장치 변수 저장 공간을 할당한 뒤 그 값을 넘겨준다. 또한 충돌 검사 결과를 저장할 $(n_1 \times n_2)$ 크기의 bool 타입 행렬 또한 장치 변수로서 GPU 메모리에 할당한다. 이 장치 변수들은 커널 함수 호출 시 매개 변수로 전달한다. 커널 함수를 호출하는 코드의 예제는 다음과 같다.

```
dim3  threads( b_wid, b_hig );
dim3  grid( T1/b_wid, T2/b_hig  );
kernel<<< grid, threads >>>(d_odata, d_v0, d_v1, d_v2, d_u0,
                            d_u1, d_u2);
```

여기서, b_wid 와 b_hig 는 각각 블록의 x 축 방향 크기와 y 축 방향 크기이고, T1 과 T2 는 각각 첫 번째 객체를 구성하는 삼각형의 개수와 두 번째 객체를 구성하는 삼각형의 개수이다. 또한, d_v0, d_v1, d_v2, d_u0, d_u1, d_u2 은 각각 삼각형들의 정점 좌표 값을 갖는 변수들이고, d_odata 는 결과값 저장을 위해 할당해둔 변수이다.

GPU 상의 커널 함수는 삼각형 집합의 충돌 검사를 담당한다. 이 때 삼각형 충돌 검사 알고리즘은 앞서 소개한 Tomas Möller 의 알고리즘에 기반을 둔다. CPU 의 호스트 프로그램이 커널을 호출하면 이 함수는 우선 매개 변수로 넘겨받은 삼각형들의 정점에 대한 정보를 공유 메모리에 저장하며, 이들은 충돌 검사 과정에서 반복적으로 사용되는 값이며 같은 블록 내의 스레드들이 공통으로 사용하는 값이기도 하므로 공유 메모리에 복사해 둔다.

커널 함수의 삼각형 충돌 검사 알고리즘이 Tomas Möller 의 알고리즘에 기반을 두고 있긴 하지만 이를 병렬처리 방식으로 재 구성 하였기 때문에 CPU 에서 시행할 때와는 달리 충돌 검사를 삼각형 조합의 개수 $(n_1 \times n_2)$ 번 만큼 반복하지 않는다. 대신 스레드의 위치를 가리키는 스레드 인덱스와 블록의 위치를 가리키는 블록 인덱스를 이용해 $(n_1 \times n_2)$ 개의 스레드가 각각 담당해야 할 한 쌍의 삼각형을 지정해준다. 이러한 삼각형 쌍들의 인덱스는 다음과 같이 계산한다.

```
int tri1 = b_hig * blockIdx.y + threadIdx.y;
int tri2 = b_wid * blockIdx.x + threadIdx.x;
```

여기서 blockIdx 는 CUDA 가 제공하는 변수로서 이 변수를 사용하는 스레드가 속한 블록의 인덱스를 값으로 가지며, 각 충돌검사 스레드는 첫 번째 객체의 tri1 번째 삼각형과 두 번째 객체의 tri2 번째 삼각형 간의 충돌 검사를 맡는다.

각 스레드는 자신이 담당한 삼각형 쌍에 대한 충돌검사가 끝나면, 결과를 저장하기 위해 할당해 둔 장치 행렬 변수에 충돌 검사 결과를 입력한다. 이 때 각 스레드는 그리드에서의 자신의 위치와 동일한 값을 인덱스로 갖는 행렬 상의 저장 공간에 결과를 입력한다. 모든 스레드들이 연산을 완료하면 시스템은 커널 함수를 종료하고 GPU 저장 공간 내의 결과 행렬 변수에 저장된 충돌 검사 값을 CPU 로 넘겨준다.

5. 구현 및 결과

본 논문에서 제안한 충돌검사 알고리즘은 C++와 CUDA 를 이용했으며, 충돌 검사를 시각적으로 확인하기 위해 사용한 그래픽스 라이브러리는 OpenGL 이다. 구현에 사용한 CPU 는 Intel Core2 Duo E6550 2.33GHz 이며, GPU 는 nVIDIA GeForce8800 GTX 를 사용하였다. 이 시스템은 CPU 로 동일한 알고리즘을 실행하였을 때에 비해 수십 배 이상의 성능 향상을 가져왔으며, 단일 부동소수점 연산(single floating point operation)만을 이용했을 경우, GPU 의 image space 상의 충돌 검사 방법들과는 달리, 충돌검사의 결과가 CPU 방식과 정확히 일치한다.

5.1 삼각형 충돌 검사 벤치마킹 시나리오

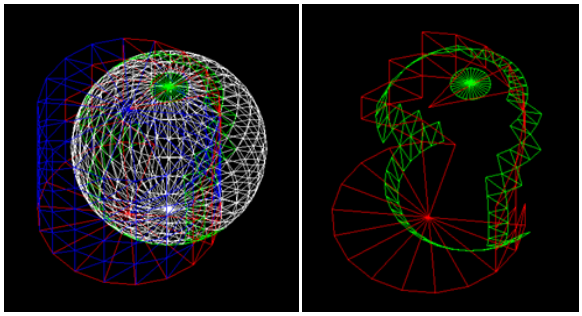
본 논문에서는 삼각형의 개수에 따른 성능 변화를 알아보고자 5 개의 테스트 모델(sphere, Cylinder, Azucar, Cup, Speen)을 준비하였다. 각 모델이 갖는 삼각형의 개수는 〈표 1〉에 나와있다. 이 다섯 개의 모델로 세 가지 시나리오를 만들었다. 첫 번째 시나리오는 Cylinder 모델과 Sphere 모델의 충돌 검사로 가능한 삼각형 쌍의 개수는 20 만 개가 되어 세 가지 시나리오 중 복잡도가 가장 낮다. 두 번째는 Azucar 모델과 Cup 모델 간의 충돌 검사인데 이 경우는 가능한 삼각형 쌍의 개수가 4 천만 개이고, 마지막 세 번째 시나리오는 Cup 모델과 Spoon 모델 간의 충돌 검사로서 가능한 삼각형 쌍의 개수가 2 억 개가 되어 가장 높은 복잡도를 갖는다.

	모델 1	모델 2
시나리오 1	960 (Sphere)	216 (Cylinder)
시나리오 2	7580 (Cup)	5250 (Azucar)
시나리오 3	7580 (Cup)	26012 (Spoon)

〈표 1〉

6.1.2 삼각형 충돌 검사 구현 결과

〈그림 4〉는 첫 번째 시나리오의 결과를 시각적으로 랜더링한 것이다.

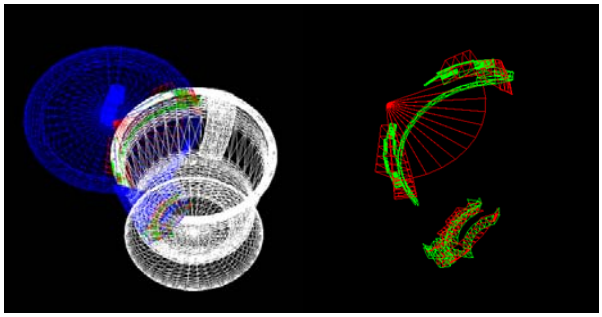


〈그림 3〉

〈그림 3〉의 왼쪽 그림은 두 개의 테스트 모델을 나타낸 것이며, 오른쪽 그림은 두 모델의 삼각형들 중 충돌을 일으키는 삼각형들만을 나타낸 것이다. 여기서, 흰색이나 파란색의 삼각형은 어떤 삼각형과도 충돌하지 않는 삼각형을 나타내며, 빨간색이나 초록색의 삼각형은 하나 이상의 삼각형과 충돌하는 삼각형이다. 이 중 GPU 를 사용한 충돌검사에만 소요된 시간은 평균 1.40ms 이며 CPU 에서 GPU 로 매개 변수 값을 전송하는 데 드는 시간은 평균 0.16ms, GPU 에서 CPU 로 결과값을 전송하는 데 드는 시간은 평균 0.78ms 이다.

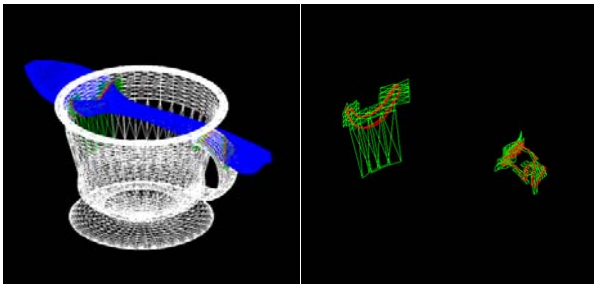
〈그림 4〉는 두 번째 시나리오의 결과이다.
이 시나리오에서 CPU 로부터 GPU 로 매개 변수 값을 전송하는 시간은 0.47ms, GPU 로부터 CPU 로 결과값을

전송하는 시간은 34.57ms, 그리고 GPU 로 충돌 검사를 하는 순수 연산 시간은 148.86ms 이다.



〈그림 4〉

〈그림 5〉는 세 번째 시나리오의 결과이다.



〈그림 5 〉

이 시나리오에서 CPU 에서 GPU 로의 매개변수 값 전송 시간은 1.03ms, GPU 에서 CPU 로의 결과값 전송 시간은 171.86ms, 그리고 충돌검사에만 소모된 시간은 909.24ms 이다. 삼각형의 개수가 많아질수록 데이터의 양이 많아지므로 CPU 와 GPU 간 데이터 전송 시간이 길어지고, 병렬 처리를 하는 멀티프로세서의 개수에도 한계가 있으므로 스레드의 개수가 늘어남에 따라 기다리게 되는 스레드들이 발생함으로 인해 GPU 연산 시간도 길어지게 되는 것이다.

6.1.3 CPU 기반 삼각형 충돌 검사 방법과의 성능 비교

	시나리오 1	시나리오 2	시나리오 3
CPU	11.84	2099.5	10377.62
GPU(CUDA)	2.34	183.9	1084.22

〈표 2〉

위의 〈표 2〉는 앞 절의 시나리오 1, 2, 3 을 CPU 버전과 CUDA 버전으로 실행하였을 때 각각의 계산 수행 시간을 비교한 것이다. 단위는 millisecond(ms)이며, 각각 10 번씩 실행시켜 얻은 수행 시간의 평균값이다. CUDA 버전의 경우엔 순수 GPU 연산 시간뿐 아니라, CPU 와 GPU 간 데이터 전송 시간도 총 수행 시간에 포함시켰다. 이 표를 보면 CUDA 를 이용한 병렬 알고리즘이 CPU 기반 순차 알고리즘에 비해 성능이 월등히 좋다는 것을 알 수 있다. 시나리오 1 의 경우 데이터 전송 시간을 합한 수행 소요 시간은 CPU 보다 약 5 배 빠르다. 또한 시나리오 2 의 경우엔 GPU 버전이 CPU 버전 보다 약 11 배 빠르고, 시나리오 3 의 경우엔 GPU 버전이 CPU 버전 보다 약 10 배 빠른 것을 확인할 수 있다. 삼각형의 개수가 늘어남에 따라 병렬 알고리즘의 연산 처리 시간도 늘어나긴 하지만, 본 실험에 사용한 GPU 의 경우 128 개의 프로세서를 이용해 충돌 검사를 병렬적으로 수행하기 때문에 같은 복잡도의 모델에 대해 순차적으로 충돌 검사를 수행하는 CPU 보다 훨씬 빠른 성능을 보여준다.

6. 결론

본 논문은 GPGPU 연산에 최적화된 프로그래밍 개발 환경인 CUDA 를 이용하여 가상 모델 간 충돌 검사를 병렬적으로 구현하였다. 이는 각각의 모델이 갖는 두 개의 삼각형 집합으로부터 가능한 모든 삼각형 쌍에 대해 충돌 검사를 수행하는 경우에, CPU 에 비해 훨씬 효율적이다.

이처럼 많은 데이터에 대해 SIMD 방식으로 처리할 수 있는 알고리즘의 경우 GPU 로 구현한다면 연산 속도를 줄여 전체적인 시스템 성능을 높일 수 있다.

참고문헌

[1] Xinyu Zhang, Young J. Kim, Interactive collision detection for deformable models using Streaming AABBs, *IEEE Transactions on Visualization and Computer Graphics*, 13(2), 2007

[2] Tomas Möller, A Fast Triangle-Triangle Intersection Test, *Journal of Graphics Tools*, 2(2), pp. 25-30, 1997

[3] nVIDIA, CUDA Programming Guide Ver.1.0

[4] Yoo-Joo Choi, Young J. Kim, Myoung-Hee Kim, Self-CD: Interactive Self-Collision Detection for Deformable Body Simulation Using GPUs, *Asian Simulation Conference* (LNCS Vol. 3398), 2005