GPU 기반의 최적화된 BVH와 R-Triangle을 이용한 옷감 시뮬레이션에서의 빠른 자기충돌 처리

문성혁[O], 김종현[*]
[O]강남대학교 소프트웨어응용학부,
[*]강남대학교 소프트웨어응용학부
e-mail: jonghyunkim@kangnam.ac.kr

Fast Self-Collision Handling in Cloth Simulations Using GPU-based Optimized BVH and R-Triangle

Seong-Hyeok Moon[O], Jong-Hyun Kim[*]
[O]School of Software Application, Kangnam University,
[*]School of Software Application, Kangnam University

● 요 약 ●

본 논문에서는 삼각형 메쉬 기반에서 옷감 시뮬레이션(Cloth simulation)에서 계산양이 큰 자기충돌(Self-collision) 처리를 GPU기반으로 가속화시킬 수 있는 방법에 대해 소개한다. CUDA기반으로 병렬 최적화하기 위해 본 논문에서는 1)재귀적으로 계산하여 충돌판정을 하는 BVH(Bounding volume hierarchy) 트리를 GPU기반에서 효율적으로 빌드, 업데이트, 트리 순회하는 방법을 제안하고, 2)삼각형 메쉬 기반에서는 중복되는 프리미티브(Primitive) 충돌검사를 최소화하기 위해 R-Triangle기법을 GPU에서 최적화 시키는 방법을 소개한다. 결과적으로 본 논문에서 제안하는 기법은 GPU 환경에서 옷감 시뮬레이션의 자기충돌과 객체충돌 처리를 빠르고 효율적으로 처리할 수 있도록 하였고, 다양한 장면에서 실험한 결과 모든 결과에서 빠른 시뮬레이션 결과를 얻을 수 있었다.

키워드: GPU(Graphics processing unit), 쿠다(Computed unified device architecture), 옷감 시뮬레이션(Cloth simulation), 자기충돌(Self-collision), 충돌처리(Collision handling), 최적화(Optimization)

I. Introduction

옷감 시뮬레이션은 최근에 가상현실이 이슈화가 되면서 패션, 게임, 의상디자이너 등 다양한 디지털 산업에 영향을 주는 대표적인 기술 중 하나이다**[x,x].** 특히, 실시간이나 인터랙티브하게 결과를 보면서 진행해야 하는 의상디자인 같은 경우 편집에 의한 시뮬레이션을 바로 보여야하기 때문에 병렬충돌 검사 기법이 필수적이다.

게임이나 영상특수효과 회사에서는 옷감 편집 툴을 대부분 마블러스 디자이너(Marvelous Designer)라는 저작도구를 사용하고 있다. 이 툴에서는 사용자가 옷감 도면을 편집하면 시뮬레이션을 통해 3차원에서 옷감 형태가 실시간으로 표현되기 때문에 산업에서 많이 활용되고 있다 (Fig. 1 참조). 하지만, 충돌 검출 및 처리의 프리미티브 단위가 버텍스(Vertex), 에지(Edge), 페이스(Face)이기 때문에 계산양이 크고, CPU기반 병렬화를 적용해도 성능이 장면 복잡도에 의존하기 때문에 사용자가 만족할만한 성능을 보여주는 것이 쉽지 않은 문제이다. 우리는 CUDA를 이용하여 BVH를 최적화 시킬 수 있는 방법과, 충돌처리 과정에서 중복 연산을 피할 수 있는 방법을 GPU 병렬화함으로써 옷감의 자기충돌이나 장애물 객체의 충돌을 빠르게 계산할 수 있는 방법을 제안한다.

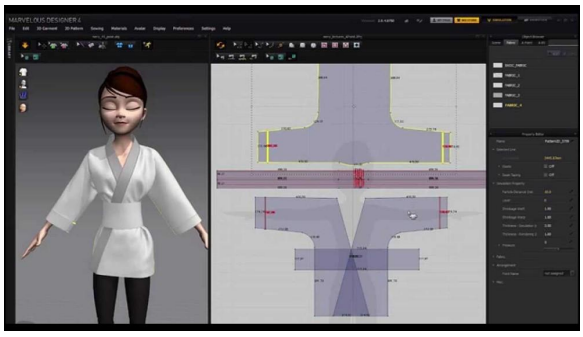


Fig. 1. Marvelous Designer in CLO Virtual Fashion Inc.

II. The Proposed Scheme

1. GPU기반 BVH 트리 구조

제한된 리소스에서 병렬화를 개선해야 하는 GPU 환경에서 BVH를 최적화하기 위해서는 다음과 같은 특징을 고려해야 한다 : 1)알고리즘의 병렬성을 살리고, 2) 메모리를 최소화하여 가속화.

트리의 업데이트나 DFS(Depth-first search)같은 트리 탐색 같은 경우 다른 레벨의 노드들은 순서에 맞게 계산해야한다. 하지만 BFS(Breadth-first search)처럼 같은 레벨의 노드들은 병렬적으로 계산할 수 있다. 본 논문에서는 이점을 최대한 활용하기 위해 같은 레벨의 노드들이 한곳에 모으는 너비 우선으로 노드들을 저장한다. 트리의 형태가 완전이진트리(Complete binary tree)라고 가정했을 때, 현재 레벨을 l, 레벨이 바뀌는 인덱스는 2^l-1로 계산할 수 있다. 또한 현재노드의 인덱스를 n이라 할 때 자식노드의 인덱스를 $2n+1, 2n+2$로 계산할 수 있다. 반대로 완전이진트리가 아니라면 노드들의 레벨이 바뀌는 인덱스의 정보와 자식 및 부모노드의 인덱스를 가지는 데이터를 따로 저장해야 한다. 따라서 메모리를 좀 더 효율적으로 최소화 할 수 있도록 BVH 트리를 항상 완전이진트리 형태로 트리를 구축했다.

1-2. GPU기반 BVH 트리 구축

BVH 트리에서 리프노드의 개수는 삼각형의 개수와 같고 완전이진트리 형태로 구축하기 때문에 삼각형의 개수만 알면 트리의 구조를 알 수 있다.

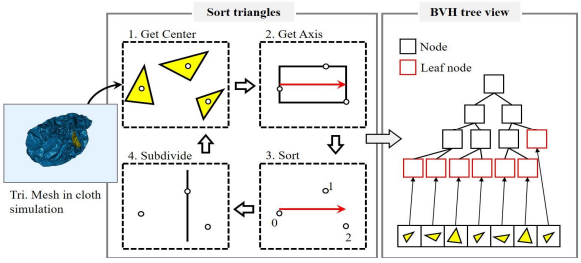


Fig. 2. Overview of building BVH tree on GPU.

따라서, Fig. 1에서 보듯이 1) 삼각형의 중심 위치에 따라 정렬하는 과정과 2) 정렬된 삼각형을 트리의 리프노드에 배치하는 두 가지 과정으로 나뉜다. 삼각형의 중심 위치에 따라 정렬하는 과정은 4가지 CUDA 커널(Kernel)함수로 세분화한다 (Fig. 2의 4가지 단계 참조). 먼저 삼각형들의 중점을 이용한 경계상자(AABB, Axis-aligned bounding box)를 찾는다. 그리고 가장 긴축을 분할 축으로 선정한다. 선정된 분할 축을 기준으로 삼각형을 정렬하고 트리의 모양에 따른 개수로 나눠준다. 이때 정렬은 GPU 환경에서 효율적이라 알려진 바이토닉 정렬(Bitonic sort)를 이용했고[x], 나눠주는 과정은 다음과 같다 (Fig. 3 참조).

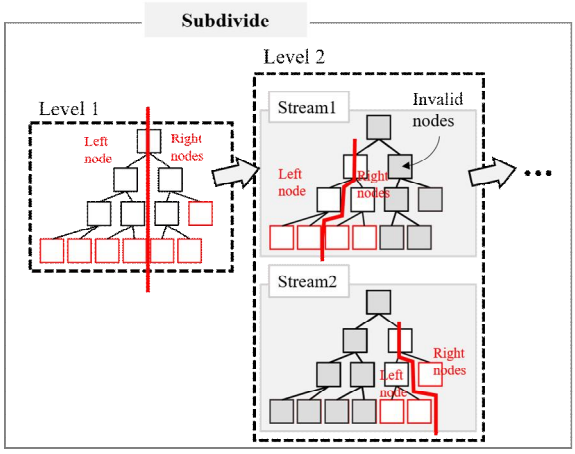


Fig. 3. Node subdivision.

트리의 최대레벨을 L, 현재 나눠지고 있는 레벨을 l, $L-l$을 $L^{'}$, 삼각형의 개수이자 리프노드의 개수를 k라 할 때 한쪽으로 나눠줘야 할 삼각형의 개수는 다음과 같다 (수식 1 참조).

$$\begin{cases} k-2^{L^{'}-2} & (k < \frac{3}{4}\cdot 2^{L}) \\ 2^{L^{'}-1} & (k \geq \frac{3}{4}\cdot 2^{L}) \end{cases} \quad (1)$$

이 과정을 반복하여 삼각형들을 나눠주며, 이 과정에서 삼각형들이 여러 묶음으로 나눠지게 된다. 이때 CUDA 스트림(Stream)을 이용하여 병렬적으로 계산해 가속화한다.

2. GPU기반 BVH 트리 업데이트 및 순환

BVH 트리의 업데이트는 리프노드부터 루트노드 순서로 AABB의 크기를 업데이트한다. 따라서 각 레벨의 노드 업데이트를 병렬적으로 계산하여, GPU 아키텍처에서 효율적일 수 있도록 설계했다.

트리의 순환은 BFS가 병렬적으로 계산되기 쉽지만, 다음 레벨에 탐색해야할 노드에 대한 데이터를 저장해야하는 과정이 필요하다. 이러한 과정은 제한적인 GPU 환경에서는 친화적이지 않기 때문에 상황 복잡도에 따라 메모리를 초과하는 경우가 발생한다. 따라서 본 논문에서는 메모리에 제한적이지 않는 CUDA기반의 DFS 방식으로 순환하는 방법을 제안한다.

GPU에서 DFS방법은 다음과 같다. 각 스레드에 위치에 맞는 리프노드의 삼각형을 배치한다 (Fig. 3 참조). 그리고 해당 삼각형에 대해 BVH 트리를 순환하며 하며 충돌검사를 하게 된다. 이때 스레드의 위치에 맞게 삼각형을 배치함으로서 BVH 트리 특성상 비슷한 위치의 삼각형이 같은 워프(Warp)에 위치하게 된다. 따라서 거의 비슷한 경로로 순환하기 때문에 병렬적으로 계산하게 되어 더 좋은 퍼포먼스를 기대할 수 있다.

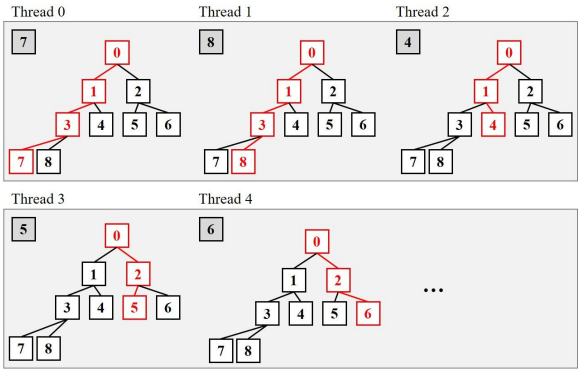
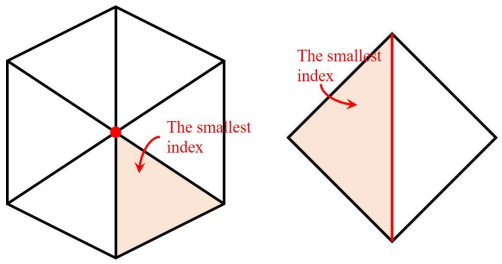


Fig. 4. DFS on GPU.

3. Representative-Triangle

메시의 구조를 보면 정점과 에지를 여러 삼각형이 공유하고 있고, 이러한 문제 때문에 충돌 검사를 할 때 계산양이 증가하게 된다. 본 논문에서는 이 문제를 R-triangle(Representative-triangle)을 통해 해결하였으며, 이 과정역시 GPU 커널에서 동작하도록 디자인하였다.



(a) Vertex (b) Edge

Fig. 5. Collision candidates selected based on the R-triangle.

R-triangle 기법에서 정점과 에지의 대표삼각형을 효율적으로 설정하는 방법이 몇 가지 제안되었다. 하지만 본 논문에서는 단순하게 공유하고 있는 삼각형 중 제일 작은 인덱스를 가지고 있는 삼각형을 대표삼각형으로 설정했다 (Fig. 5 참조).

III. Results

Fig. 6은 본 논문에서 제안하는 GPU기반 BVH 트리와 기존의 CPU기반 BVH 트리를 비교한 결과이다. 옷감을 다양한 해상도로 표현하고, 자기충돌이 많이 발생하는 장면을 구성하여 결과를 비교했다. Table 1~3은 BVH 트리의 빌드시간, 업데이트시간 그리고 탐색시간이다.

(a) Low resolution (b) Medium resolution (c) High resolution

Fig. 6. Self-collision handling with our method.

Table 1. BVH tree build

# of tri.	GPU (ours)	CPU
2,244	23.94 ms	5.71 ms
27,030	59.12 ms	45.18 ms
35,992	0.21 s	0.98 s

Table 2. BVH tree update(refit)

# of tri.	GPU (ours)	CPU
2,244	0.16 μs	0.31 μs
27,030	0.18 μs	1.28 μs
35,992	0.35 μs	4.81 μs

Table 3. BVH tree search(traverse)

# of tri.	GPU (ours)	CPU
2,244	1.19 ms	10.28 ms
27,030	12.52 ms	162.92 ms
35,992	0.41 s	8.65 s

옷감 메쉬의 삼각형 개수가 3만개 미만일 때 메모리 할당 시간 때문에 GPU가 CPU보다 느리다. 하지만 3만개 이상부터 5~10배 속도가 향상되는 결과를 얻었다 (Table 1 참조). BVH 트리의 업데이트는 메모리 할당이 필요 없고 커널함수만 실행된다. 또한 병렬적으로 계산이 되기 때문에 계산 량이 적음에도 불구하고 굉장히 좋은 효율을 보여주고 있다. CPU보다 최대 10배 이상의 속도가 향상되었다 (Table 2 참조). 트리 탐색시간은 너비 우선탐색이 아닌 깊이 우선 탐색 방법으로 탐색하기 때문에 병렬성이 떨어지지만 그럼에도 불구하고 10~20배 속도가 향상되는 높은 효율을 보여주었다.

IV. Conclusions

본 논문에서는 GPU 환경에 맞게 충돌검사를 위한 BVH 트리를 효율적으로 빌드하고 순환하는 방법을 제안했다. 이 기법은 병렬성을 최대한 살리고 가속화 하는 것에 초점을 맞췄다. 하지만 순환하는 과정에서 병렬성과 메모리를 동시에 해결하는 방법을 찾아내지 못했다. 향후 메모리 문제를 해결하여 너비 우선 탐색 방식으로 충돌검사를 할 수 있도록 확장할 예정이다.

REFERENCES

[1] Müller, Matthias, et al. "Position based dynamics." Journal of Visual Communication and Image Representation, Vol. 18, No. 2, pp. 109-118, 2007.

[2] Muller, Matthias, N. Chentanez, T. Y. Kim, and M. Macklin. "Strain based dynamics." In Eurographics/ACM SIGGRAPH Symposium on Computer Animation, pp. 2, 2014.

[3] Choi, Kwang-Jin, and Hyeong-Seok Ko. "Stable but responsive cloth." ACM SIGGRAPH, 2005.

[4] Bridson, Robert, Sebastian Marino, and Ronald Fedkiw. "Simulation of clothing with folds and wrinkles." ACM SIGGRAPH 2005, 2005.

[5] Kang, Young-Min, and Hwan-Gue Cho. "Complex deformable objects in virtual reality." In Proceedings of the ACM symposium on Virtual reality software and technology, pp. 49-56, 2002.