공학석사 학위논문

변형 가능한 모델의 충돌 탐지를 위한 효율적인 법선 원뿔 컬링 방법

Efficient normal cone culling method

for collision detection of deformable model

2020년 8월

서울대학교 대학원

전기 · 정보공학부

이 창 진

변형 가능한 모델의 충돌 탐지를 위한 효율적인 법선 원뿔 컬링 방법

Efficient normal cone culling method

for collision detection of deformable model

지도 교수 고 형 석

이 논문을 공학석사 학위논문으로 제출함

2020년 5월

서울대학교 대학원

전기 · 정보공학부

이 창 진

이창진의 공학석사 학위논문을 인준함

2020년 5월

위 원 장 최 진 영 (인)

부위원장 고 형 석 (인)

위 원 김 영 민 (인)

초 록

강체 시뮬레이션과 달리 변형이 가능한 물체의 시뮬레이션 (직물 시뮬레이션)에서는 같은 물체 안에서 모든 삼각형 쌍의 충돌 탐지를 해야 하므로 매우 많은 시간이 소모된다. 이러한 이유로 현재 그래픽기술과 하드웨어를 이용하여 실시간으로 사실적인 물리 기반 직물 시뮬레이션을 만들어 내는 데 많은 어려움을 겪고 있다.

이에 따라 변형 가능한 물체 시뮬레이션에서의 충돌 탐지 시간을 줄이기 위해서 많은 가속화 연구들이 진행되어 왔다. 본 논문에서 제안하는 방법은 충돌 탐지 시간을 줄이기 위한 연구 중 직물의 표면 법선 벡터를 이용하여 충돌 탐지 연산을 줄이는 방법에서 영감을 받아 진행되었다.

본 논문에서는 기존에 제시된 표면 법선 벡터를 이용하는 방법에서 나아가 좀 더 효율적으로 표면 법선 벡터를 이용하여 충돌 탐지 시간을 줄이는 방법을 제시한다. 본 논문에서 제시하는 방법은 불연속 충돌 탐지에도 적용이 가능하고 연속 충돌 탐지에도 이용이 가능하다. 또한 연속 충돌 탐지에 적용할 때는 특히 동적인 장면에서 본 논문에서 제시한 방법이 좋은 효율을 보인다.

결론적으로 논문에서 소개한 방법은 기존의 방법에 비해 충돌 검사를 진행해야 하는 삼각형 쌍의 개수가 10~40% 감소하는 결과를 보였다. 본 논문의 방법은 이진 트리 구성하는 시간에서 트레이드 오프가 발생하나 전체적으로 봤을 때 성능 향상이 있다.

주요어 : 법선 원뿔, 충돌 탐지, 직물 시뮬레이션, 연속 충돌 탐지, 불연속 충돌 탐지
학 번 : 2018-26787

목 차

표 목차

그림 목차

제 1 장 서론

옷과 같은 변형이 가능한 모델에 관한 시뮬레이션은 상당히 오래 전부터 연구가 이루어지던 분야이다. 변형 가능한 모델을 실제적으로 시뮬레이션 하기 위해서 삼각형 단위의 충돌 처리가 필수적이다. 하지만 모든 삼각형 쌍의 충돌 검사를 진행하는 것은 상당히 많은 계산양을 요구한다. 이에 따라 충돌 탐지 연산을 최소화하여 성능을 향상시키는 많은 방법들이 연구되어왔다.

변형 가능한 모델의 충돌 처리는 크게 두 가지로 나눌 수 있다. 특정 순간에서의 충돌 여부를 확인하는 불연속 충돌 처리와 특정 시간 구간 안에서의 충돌 여부를 확인하는 연속 충돌 처리가 있다.

1994년 Volino와 Thalmann[1]은 처음으로 표면 법선을 이용하여 충돌 처리 가속화를 할 수 있음을 제시하였다. 1997년 Provot[2]은 표면 법선을 이용하여 법선 원뿔(normal cone)을 형성하고 이를 통해 같은 천 안에서의 불연속 충돌 처리를 가속화하는 방법을 제시하였다. 이 후 2009년 Tang[3]은 법선 원뿔 방법을 확장하여 연속 충돌 처리에 적용하였다. 법선 원뿔 방법을 이용해 상당히 많은 거짓-긍정(false-positive) 충돌을 걸러낼 수 있었고 많은 가속화를 이루었다.

하지만 기존에 제시된 법선 원뿔을 구성하는 방법은 표면 곡률을 이용하여 충돌 처리 연산을 줄이는 최적화된 방법은 아니다. 거짓-긍정 충돌을 상당히 많은 양 거를 수 있음에도 불구하고 아직 많은 양의 거짓-긍정 충돌이 존재함을 확인하였다. 본 논문에서는 좀 더 효율적인 법선 원뿔 구성법을 제시함으로써 더 많은 거짓-긍정 충돌을 걸러낼 수 있음을 보일 것이다.

제 1 절 용어 설명 (Terminology)

본 논문의 서술이나 이해를 쉽게 하기 위해 본 논문에서 사용되는 약어나 용어를 정의하도록 하겠다. V, E, F, T는 각각 점, 선, 면, 삼각형을 의미한다. $\boldsymbol{n}$은 특정 법선 벡터(normal vector)를 의미한다. 본 논문에서 연속 충돌 탐지(continuous collision detection, CCD)를 서술할 때, 시간 간격 $t \in [0,1]$으로 한다. 따라서 $\boldsymbol{n}_t^i$ 와 같은 표현을 사용한다면 i번째 삼각형의 특정 시간 t ($t \in [0,1]$)에서의 법선 벡터를

의미한다. 단순히 $\boldsymbol{n}_t$ 와 같이 표현한다면 기술하고자 하는 삼각형이 분명 할 때, 그 삼각형의 특정 시간 t ($t \in [0,1]$)에서의 법선 벡터를 의미한다. 가령 $\boldsymbol{n}_0^i$, $\boldsymbol{n}_1^i$ 로 표현했다면, 각각 i 번째 삼각형의 이산적 시간(discrete time) $t = 0$에서의 법선 벡터와, 이산적 시간 $t = 1$에서의 법선 벡터를 의미한다. 본 논문에서 불연속 충돌 탐지(discrete collision detection, DCD)를 서술할 때, 별도의 시간 t를 표시하지 않는다면 특정 이산적 시간을 의미한다. 가령 불연속 충돌 탐지를 서술하는 부분에서 $\boldsymbol{n}_i$ 로 표현한다면 이산적 시간에서의 특정 삼각형 i 의 법선 벡터를 의미한다.

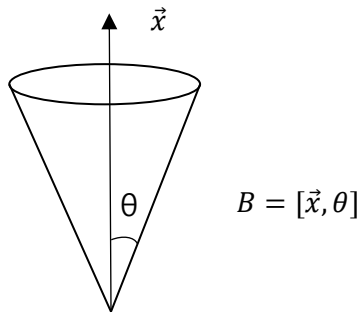


그림 1−1. $\vec{x}$를 중심축으로 하고, 중심축 각도가 θ인 법선 원뿔

법선 원뿔(normal cone) $B = [\vec{x}, \theta]$에서 $\vec{x}$는 법선 원뿔의 중심축을 의미하고 θ는 중심축 각도를 의미한다. 이 때 중심축은 항상 단위 벡터(unit vector)를 의미한다(그림 1−1).

제 2 장 관련 연구

변형 가능한 모델(deformable model)의 시뮬레이션은 오랜 기간 컴퓨터 그래픽스 분야에서 연구되어 왔다. 변형 가능한 모델을 시뮬레이션 할 때 많은 연산이 수행되는 부분은 시뮬레이션 부분과 충돌 처리(collision handling) 부분이다. 시뮬레이션 부분과 충돌 처리 부분에서는 각각 가속화를 위해 많은 연구가 진행되어 왔다.

시뮬레이션 부분에서 물리적인 접근 방식과 암묵적 오일러 방식을 이용하여 1998년 Baraff와 Witkin[4]는 large time step에서 시뮬레이션을 가능케 하였다. 2002년 Choi와 Ko[5]는 2단계 역오일러 방법을 사용하여 기존 불안정한 물리 시뮬레이션에 안정성을 더하였다. 2013년 Tang[6]은 GPU를 물리 시뮬레이션에 효율적으로 이용하기 위한 연구를 진행하였다. 또한 수학적인 multigrid 방법을 이용하여 시뮬레이션을 가속화하는 연구도 계속해서 진행되었다(2015년 Rasmus Tamstorf[7]).

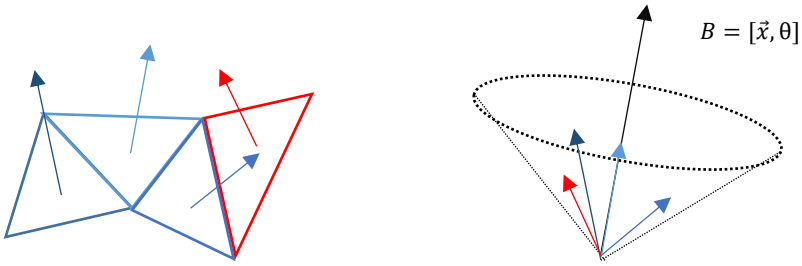
충돌 처리 부분에서는 다양한 방법으로 충돌 처리 가속화 연구가 진행되었다. bounding box를 효율적으로 구성하여 가속화하는 연구가 진행되었다. 2009년 Tang[3]은 모델의 전체적인 연결관계를 활용하여 충돌 처리를 가속화하였다. 2011년 Tang[8]은 GPU를 활용하여 충돌 처리를 가속화하였다.

기본적으로 변형 가능한 모델에서의 충돌 처리 과정은 서로 다른 물체 사이의 충돌과 한 물체 안에서의 충돌로 분류할 수 있다. 한 물체 안에서의 충돌은 자기 충돌(self-collision)이라고 명명되어왔다. 변형 가능한 물체를 시뮬레이션 할 때 자기 충돌은 상당히 많은 계산양을 요구하게 된다.

자기 충돌에 필요한 많은 계산양을 줄이기 위해 다양한 연구들이 진행되어 왔다. 본 논문에서는 다양한 연구 중 법선 원뿔에 관련된 연구를 중심적으로 다루고 발전시켰다.

제 1 절 불연속 충돌 탐지에서의 법선 원뿔 컬링 방법 (Normal cone culling method for discrete collision detection)

1994년 Volino와 Thalmann[1] 자기 충돌이 일어날 수 없는 두가지 조건을 제시하였다. 첫 번째는 법선 검사(normal test)이며, 두 번째는 윤곽 검사(contour test) 이다. 첫 번째 조건은 1997년 Provot[2]에서 법선 원뿔이라는 개념을 적용하면서 효율적인 검사가 가능해졌다.



(a) 특정 지역 삼각형들과 법선 (b) (a)의 삼각형 법선을 포함하는 원뿔

그림 2-1. 특정 지역 삼각형들의 법선을 포함하는 원뿔

1997년 Provot[2]은 한 물체 안에서 표면들이 이루는 곡률이 충분히 작다면, 자기 충돌이 일어날 수 없다는 생각에 기인하여 법선 원뿔 컬링 방법을 고안하였다. 특정 부분의 곡률은 그 부분을 구성하는 법선들의 집합으로 표현될 수 있으며, 법선들을 포함하는 원뿔을 계산하였다. 그림 2-1과 같이 법선들을 포함하는 원뿔을 법선 원뿔(normal cone)이라 명하고, 법선 원뿔의 중심축과 중심축 각도를 계산하였다. 이 때 만약 법선 원뿔의 중심축 각도 $\theta < \frac{\pi}{2}$ 이면, 법선 원뿔을 구성하고 있는 부분(zone)의 곡률은 충분히 작다고 판단되고 따라서 그 부분에서는 자기 충돌이 일어나지 않는다(그림2-1).

모든 삼각형은 특정 이산 시간에 하나의 법선 벡터를 가지고 있다. 삼각형의 법선 벡터들을 이용해 계층적으로 이진 트리를 구성하여 계층적인 법선 원뿔 트리를 완성한다. 이산 시간에서 특정 삼각형의 하나의 법선 벡터는 법선 원뿔 트리의 잎 노드(leaf node)를 구성한다. 예를 들어, 삼각형 T_i 는 특정 이산 시간에서 법선 벡터 $\boldsymbol{n}_i$ 를 지니고 있다. 이것은 $B_i = [\boldsymbol{n}_i, 0]$ 법선 원뿔로 나타낼 수 있다. 하나의 법선 벡터이므로 중심축 각도는 0이 된다. 법선 원뿔 트리에서 $B_i = [\boldsymbol{n}_i, 0]$은 잎 노드 법선 원뿔을 구성한다.

법선 원뿔 트리는 상향식(bottom-up)으로 계층적 이진 트리를 구성한다. 따라서 잎 노드가 모두 구성되었다면 식(1) 병합 방법으로 이용해 병합을 하여 부모 노드 법선 원뿔을 만들어 나간다 (그림 2-2). 이를 상향식 방식으로 반복하여 전체적인 법선 원뿔 트리를 완성한다.

$$\begin{cases} \vec{x} = \dfrac{\overrightarrow{x_1} + \overrightarrow{x_2}}{2} \\ \beta = \cos^{-1}(\overrightarrow{x_1} \cdot \overrightarrow{x_2}) \\ \alpha = \dfrac{\beta}{2} + \max(\alpha_1, \alpha_2) \end{cases} \tag{1}$$

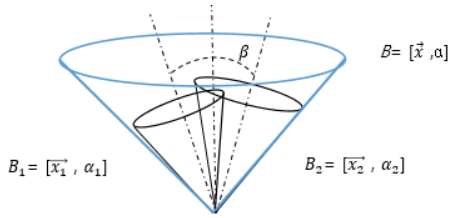


그림 2-2. 두 개의 자식 노드 법선 원뿔을 이용하여 구성된 부모 노드 법선 원뿔

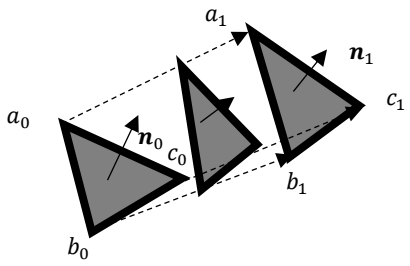
제 2 절 연속 충돌 탐지에서의 법선 원뿔 컬링 방법 (Normal cone culling method for continuous collision detection)

2009년 Tang[3]은 1997년 Provot[2]에서 제안한 불연속 충돌 탐지에서의 법선 원뿔을 확장하여 연속 충돌 탐지에서의 법선 원뿔을 제안하였다. 이를 통해, 연속 충돌 탐지에 법선 원뿔 컬링 기법을 적용함으로써 시뮬레이션의 충돌 탐지에서 매우 많은 가속화를 이루어 냈다.

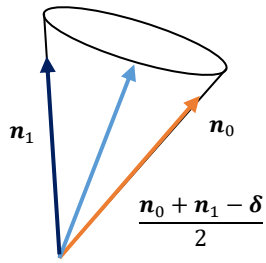
$$\boldsymbol{\delta} = (\overrightarrow{\boldsymbol{v_b}} - \overrightarrow{\boldsymbol{v_a}}) \times (\overrightarrow{\boldsymbol{v_c}} - \overrightarrow{\boldsymbol{v_a}}) \tag{2}$$

$$\boldsymbol{n_t} = \boldsymbol{n_0} + (\boldsymbol{n_1} - \boldsymbol{n_0} - \boldsymbol{\delta}) \cdot \boldsymbol{t} + \boldsymbol{\delta} \cdot \boldsymbol{t^2} \tag{3}$$

$$\boldsymbol{n_t} = \boldsymbol{n_0} \cdot (\boldsymbol{1} - \boldsymbol{t})^2 + \frac{(\boldsymbol{n_0} + \boldsymbol{n_1} - \boldsymbol{\delta})}{\boldsymbol{2}} \cdot \boldsymbol{2t} \cdot (\boldsymbol{1} - \boldsymbol{t}) + \boldsymbol{n_1} \cdot \boldsymbol{t^2} \tag{4}$$



(a) 삼각형의 $t \in [0,1]$에서 움직임
$\overrightarrow{v_a} = a_1 - a_0$, $\overrightarrow{v_b} = b_1 - b_0$,
$\overrightarrow{v_c} = c_1 - c_0$



(b) 삼각형의 연속 법선 원뿔

그림 2-3. $t \in [0,1]$에서 한 삼각형의 법선 n_t를 법선 원뿔로 표현한 것.

시뮬레이션 시간 간격 $t \in [0,1]$ 라고 할 때, 식(3)의 n_t 는 $t \in [0,1]$ 에서 삼각형의 법선 벡터를 의미한다. 2009년 Tang[3]에서는 식(3)과 같이 n_t 를 베지어 곡선(Bezier curve) 모양으로 전개가 가능함을 보였다. 또한 베지어 곡선의 볼록 껍질(convex hull) 특성에 기반하여 n_t가 항상 세 개의 제어점(control points) n_0, $\frac{n_0+n_1-\delta}{2}$, n_1 안으로 들어온다는 점을 이용하였다. $t \in [0,1]$일 때 n_0, $\frac{n_0+n_1-\delta}{2}$, n_1 가 이루는 볼록 껍질 안으로 n_t가 항상 들어가기 때문에 n_0, $\frac{n_0+n_1-\delta}{2}$, n_1을 이용하여 법선 원뿔을 만들어서 n_t을 하나의 법선 원뿔로 표현해 냈다.

따라서 그림 2-3과 같이 시간 $t \in [0,1]$ 일 때, 하나의 삼각형에 대해 원뿔을 형성하고 그것은 법선 원뿔 트리(normal cone tree)에서 하나의 잎 노드(leaf node)를 형성한다. 이 때, 2009년 Tang[3] 은 n_0, $\frac{n_0+n_1-\delta}{2}$, n_1 을 가지고 하나의 법선 원뿔(normal cone)의 중심축(axis)과 중심각(apex angle)을 이루기 위해서 기존 1997년 Provot[2]에서 제시한 병합 방법(merging method)을 채택하였다. n_0, $\frac{n_0+n_1-\delta}{2}$, n_1 3개의 벡터 중 두 벡터를 병합시켜 하나의 원뿔을 만들고 만들어진 원뿔과 나머지 하나의 벡터에 병합 방법을 적용하여 하나의 잎 노드 법선 원뿔을 만들었다.

만들어진 잎 노드 법선 원뿔들을 활용해 계층적인 법선 원뿔 트리를 만들 때는 1997년 Provot[2]이 제시한 병합 방법을 그대로 이용하였다.

제 3 절 3차원 벡터들을 감싸는 최적의 경계 원뿔 (Optimal bounding cones of vectors in three dimensions)

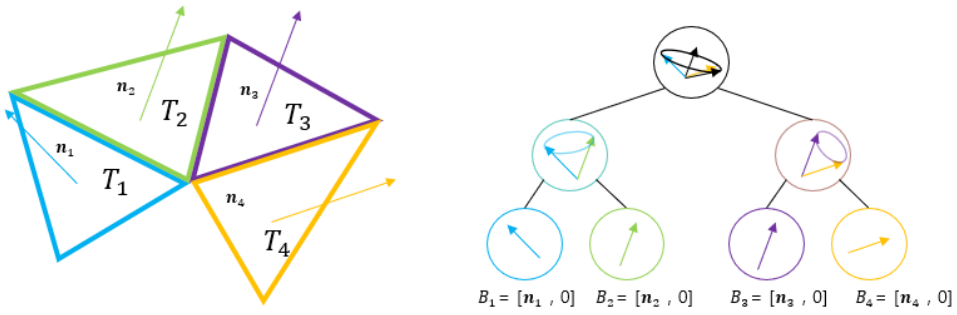
3차원 벡터 집합을 감싸는 최소-각도의 원뿔(minimum-angle bounding cone)을 구하는 문제는 컴퓨터 그래픽스 분야뿐만 아니라 기하학적 모델링 등에서 많은 적용이 가능하다. 따라서 오랜 시간 동안 이 문제에 관한 연구가 진행되어 왔다. 2005년 Barequet[9]은 이러한 문제에 관해 최적의 경계 원뿔(optimal bounding cone)을 구성하는 알고리즘을 제안하였다.

제 3 장 개요

이 부분에서는 본 논문에서 제시하는 법선 원뿔 방법이 불연속 충돌 탐지와 연속 충돌 탐지에서 어떻게 이용되는지 개요를 설명하겠다.

우선 앞서 말했듯이 법선 원뿔은 불연속 충돌 탐지와 연속 충돌 탐지의 연산 최소화를 위한 거르기 방법(culling method)이다. 법선 원뿔을 적용하기 위해서는 크게 두가지 과정이 필요하다. 첫 번째로 전체 메쉬(mesh)에 대한 법선 원뿔 트리 형성이 필요하다. 두 번째로 형성된 법선 원뿔 트리를 탐색하며 충돌 검사를 하는 과정이 필요하다.

제 1 절 법선 원뿔 트리 형성 방법 (Normal cone tree construction method)



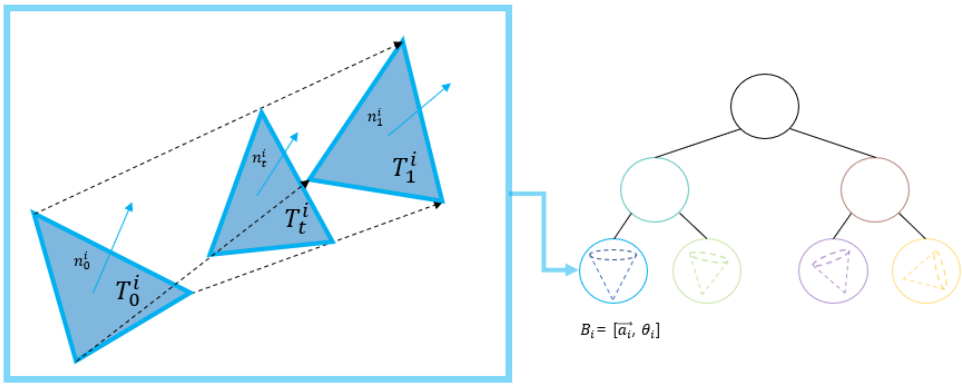
(a) 특정 이산 시간에서 삼각형들과 법선 벡터

(b) (a) 삼각형들에 의해 구성된 이진 법선 원뿔 트리

그림 3-1. (a)는 특정 이산 시간에서의 네 개의 삼각형을 나타낸 모습이며 (b)는 왼쪽 삼각형들의 법선 벡터들을 이용해 구성한 법선 원뿔 트리이다.

앞서 말했듯이 불연속 충돌 탐지는 이산적 시간에 관해서 충돌이 있는지 없는지 검사를 실행하는 것이다. 그림 3-1 (a)와 같이 특정 삼각형 T_i은 특정 이산 시간에서 하나의 법선 벡터 $\boldsymbol{n}_i$를 가지고 있다. 각 삼각형의 법선 벡터 $\boldsymbol{n}_i$는 그림 3-1 (b)와 같이 이진 트리의 잎 노드(leaf node)를 각각 하나씩을 구성한다. 잎 노드를 다 구성한 후에는 각각 두 개의 자식 노드를 병합하여 한 개의 부모 노드들을 구성한다. 그림 3-1 (b)에서 보이듯 상향식(bottom-up)으로 계속해서 병합 방법을 이용하여 뿌리 노드(root node)까지 전체적인 법선 원뿔 트리를 구성한다. 지금까지 법선 원뿔에 관해서 연구된 대부분의

논문에서는 1997년 Provot[2]가 제시한 병합 방법(식 (1))을 이용하였다. 본 논문에서는 새로운 병합 방법을 제시할 것이다.



(a) $t \in [0,1]$ 시간 간격에서 i번째 삼각형의 이동 모습

(b) $t \in [0,1]$ 시간 간격에서 i번째 삼각형의 법선 벡터를 감싼 법선 원뿔은 법선 원뿔 트리에서 하나의 잎 노드를 형성한다.

그림 3-2. (a)는 $t \in [0,1]$ 시간 간격에서 i번째 삼각형과 법선 벡터의 변화 모습을 나타낸 것이며 (b)는 전체 법선 원뿔 트리를 나타낸다.

연속 충돌 탐지는 특정 시간 간격(time step) 안에서 충돌 존재 여부를 찾는 것이다. 따라서 특정 시간에서 변화하는 법선 벡터를 하나의 법선 원뿔로 감싸서 표현해야 한다. $t \in [0,1]$ 에서 변화하는 $\boldsymbol{n}_t^i$는 하나의 법선 원뿔로 표현될 수 있고, 전체 법선 원뿔 트리에서 하나의 잎 노드(leaf node) 법선 원뿔을 구성하게 된다. 2009년 Tang[3]은 $\boldsymbol{n}_t^i$ ($t \in [0,1]$)을 하나의 잎 노드 법선 원뿔로 감싸기 위한 방법을 제시하였다. 본 논문에서는 2009년 Tang[3]이 제시한 방법을 발전시키고 2005년 Barequet[9]이 제시한 알고리즘을 적용하여 좀 더 세밀한 잎 노드 법선 원뿔 구성 방법을 제시할 것이다.

제 2 절 법선 원뿔 트리 이용한 컬링 방법 (Culling method using normal cone tree)

.

전체적인 법선 원뿔 트리를 구성하였다면 충돌 탐지 가속화를 위해 법선 원뿔 트리를 이용하는 과정이 필요하다. 이 부분에서는 간단히 법선 원뿔을 이용하는 방법에 대해서 정리를 하고 넘어가도록 하겠다. 법선 원뿔 트리를 이용할 때는 하향식(top-down)으로 순회를 진행한다.

하향식으로 각 노드(node)의 법선 원뿔 $B = [\vec{x}, \theta]$ 의 중심 각도를 확인한다. 만약 중심 각도 $\theta < \frac{\pi}{2}$ 라면, 이 구간 안에서는 자기 충돌이 없다고 판단할 수 있다. 따라서, $\theta < \frac{\pi}{2}$ 을 만족시키는 노드의 자식 노드들은 더 이상 탐색을 진행하지 않는다.

제 4 장 법선 원뿔 구성 방법

1997년 Provot[2]의 법선 원뿔 병합 방법은(식(1)) 두 자식 원뿔을 모두 감싸 부모 원뿔을 만들 수 있다는 점에서는 의심할 여지가 없다. 그러나 두 원뿔을 병합하는 과정에서 포함되지 않아도 될 부분이 법선 원뿔에 들어옴으로써 전체 트리를 순회하며 컬링(culling)을 진행할 때 효율이 떨어지게 된다. 예를 들어 두 개의 자식 법선 원뿔 $B_1 = [\overrightarrow{x_1}, \alpha_1]$, $B_2 = [\overrightarrow{x_2}, \alpha_2]$을 이용해 부모 노드의 법선 원뿔 $B = [\vec{x}, \alpha]$을 구성한다고 할 때, 1997년 Provot[2]이 제시한 방법에선 법선 원뿔 $B = [\vec{x}, \alpha]$의 중심축 각도 α는 $\overrightarrow{x_1}$, $\overrightarrow{x_2}$ 의 사잇각에 α_1와 α_2 중 더 큰 각도를 더함으로써 구한다. 이 때, α_1와 α_2 중 더 큰 각도를 더하는 과정에서 α_1와 α_2의 각도 차이가 크다면 α 값이 불필요하게 커지게 될 것이다 (그림 4-1).

따라서 본 논문에서는 새로운 병합 방법을 이용하여 불필요한 부분이 병합 과정에서 법선 원뿔로 들어오는 것을 막고 그에 따라 컬링 효율이 좋아짐을 보일 것이다.

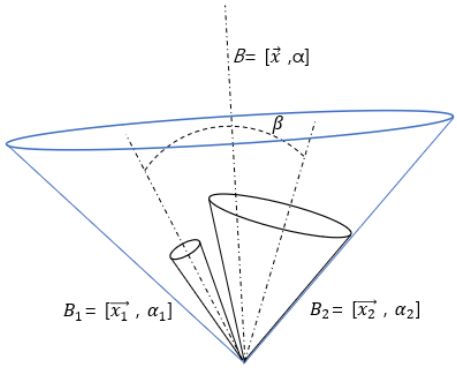
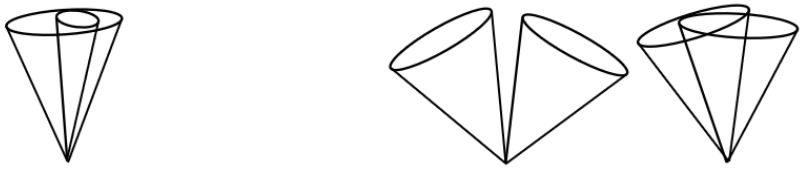


그림 4-1. 불필요한 부분도 포함되어 형성된 부모 노드 법선 원뿔

제 1 절 병합 방법 (Merging method)

먼저 두 개의 원뿔의 관계를 본 논문에서는 두 가지 경우로 나눠서 생각할 것이다(그림 4-2).



(a) 한 쪽 원뿔이 다른 쪽 원뿔에 완전히 포함되는 경우　　(b) (a) 경우가 성립하지 않는 경우

그림 4-2. 두 개의 원뿔의 관계

본 논문에서 제시하는 법선 원뿔 병합 알고리즘은 다음과 같다(알고리즘 1).

알고리즘 1. Merging method($B_{left\_child}[\overrightarrow{x_1}, \alpha_1]$, $B_{right\_child}[\overrightarrow{x_2}, \alpha_2]$)

```
1:  Merging method(B_left_child[x1, α1], B_right_child[x2, α2]) :
2:      β = cos^-1(x1 · x2)
3:      α = (β + α1 + α2 )/2
4:      weight_left = α − α1
5:      weight_right = α − α2
6:          if weight_left ≤ 0
7:              x = x1
8:              α = α1
9:          else if weight_right ≤ 0
10:             x = x2
11:             α = α2
12:         else
13:             x = sin(weight_right) * x1 + sin(weight_left) * x2
14:             x = normalize(x)
15:         end if
16:     return B_parent[x, α]
```

알고리즘 1. Merging method를 자세히 살펴보도록 하자. β는 두 자식 노드 법선 원뿔 중심축 사잇각을 의미한다. (앞서 말했듯이 모든 법선 원뿔의 중심축 벡터는 단위 벡터이다.) 만약 두 법선 원뿔의 관계가 그림 4-2 (b)와 같다면 알고리즘 1 줄 3에서 구한 각도(α)가 부모 노드 법선 원뿔의 중심축 각도가 될 것이다. 만약 두 법선 원뿔의 관계가 그림 4-2 (a)와 같다면 알고리즘 1 줄 3에서 구한 각도(α)는 두 법선 원뿔 사이의 관계를 결정짓는 데에만 이용될 것이다. 알고리즘

1의 줄 4, 5를 보면 각각$weight_{left}$와$weight_{right}$를 구한다. $weight_{left}$ 또는 $weight_{right}$가 0보다 작거나 같아진다는 것은 그림 4-2 (a)관계를 가진다는 의미이고, 따라서 부모 노드 법선 원뿔은 더 작은 원뿔을 포함하는 법선 원뿔로 유지하게 된다.

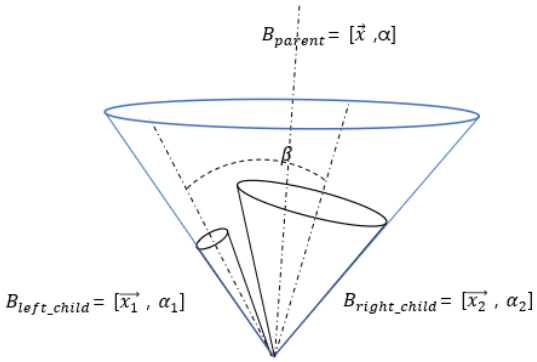


그림 4-3. 알고리즘 1을 통해 형성된 부모 노드 법선 원뿔

$weight_{left}$와 $weight_{right}$ 가 둘 다 양수라면 그림 4-2(b) 경우가 될 것이다. 그림 4-2(b)의 경우라면 알고리즘 1 줄 13과 같은 방법으로 부모 노드 법선 원뿔의 중심축을 구할 수 있다. 이것은 앞서 구한 법선 원뿔들의 중심축 벡터가 모두 단위 벡터이므로 내분의 성질을 이용하여 만들어진 식이다.

본 논문에서는 알고리즘 1의 병합 방법을 이용하여 전체적인 법선 원뿔 트리를 상향식(bottom-up)으로 구성하였다. 기존 1997년 Provot[2]이 제시한 병합 방법과 비교했을 때 구성 시간 자체는 더 오래 걸릴 수 있지만 컬링 효율을 늘림으로써 전체적으로 시간적 이득을 얻을 수 있다. 이 방법은 불연속 충돌 법선 원뿔 트리를 구성할 경우뿐만 아니라 연속 충돌 법선 원뿔 트리를 형성할 경우에도 이용할 수 있다.

제 2 절 효율적인 연속 충돌 법선 원뿔 구성 방법(Efficient method for constructing continuous collision normal cone)

연속 충돌 탐지에 법선 원뿔을 이용하여 속도를 가속화하기 위해 2009년 Tang[3]은 베지어 곡선의 볼록 껍질 특성(convex hull property)를 이용하였다. 본 논문에서는 De casteljau’s 알고리즘을

이용하여 좀 더 세밀한 껍질을 구성하여 법선 원뿔을 형성함으로써 컬링 효율을 증가시킬 것이다. $t \in [0,1]$ 에서 특정 삼각형의 법선 벡터 변화 경로를 법선 원뿔로 감싸기 위한 기본 전개는 2009년 Tang[3]의 방식을 그대로 이용할 것이다.

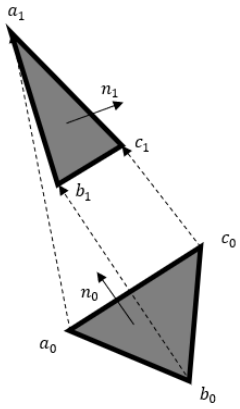
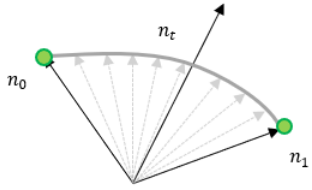
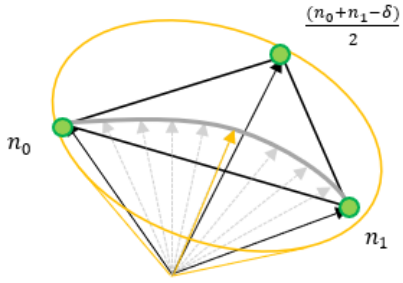


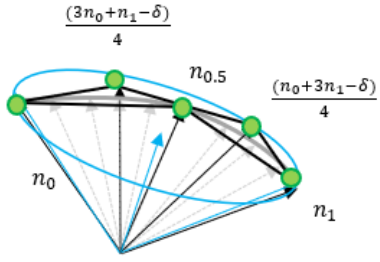
그림 4-4. $t \in [0,1]$ 시간 간격에서 점 a, b, c가 이루는 삼각형의 이동 경로와 그에 따른 법선 벡터의 변화 모습이다.



(a) $t \in [0,1]$에서 법선 벡터의 변화 경로



(b) 3개의 제어점을 이용해 만든 연속 법선 원뿔(2009년 Tang[3]이 제시한 방법)



(c) 5개의 제어점을 이용해 만든 연속 법선 원뿔

그림 4-5. (a) 법선 경로,
(b) 법선 경로를 3개의 제어점으로 감쌌을 경우,
(c) 법선 경로를 5개의 제어점으로 감쌌을 경우.

본 논문에서는 $t \in [0,1]$ 에서 $\overrightarrow{v_a} = a_1 - a_0$, $\overrightarrow{v_b} = b_1 - b_0$, $\overrightarrow{v_c} = c_1 - c_0$ 와 같이 정의한다. 따라서 $t \in [0,1]$에서 각 점의 위치는 $a_t = a_0 + \overrightarrow{v_a} \cdot t$, $b_t = b_0 + \overrightarrow{v_b} \cdot t$, $c_t = c_0 + \overrightarrow{v_c} \cdot t$ 로 표현할 수 있다. 또한 전개의 편의를 위해 $\boldsymbol{\delta} = (\overrightarrow{v_b} - \overrightarrow{v_a}) \times (\overrightarrow{v_c} - \overrightarrow{v_a})$ 로 정의한다.

2009년 Tang[3]에 의하면 점의 위치가 시간 t 에 관해서 선형 보간(linear interpolation)으로 구해진다고 할 때, $t \in [0,1]$에 관한 법선 벡터는 t에 관한 2차식으로 표현된다(식 (5)).

$$\boldsymbol{n_t = n_0 + (n_1 - n_0 - \delta) \cdot t + \delta \cdot t^2} \tag{5}$$

2009년 Tang[3]은 식(5)를 전개하여 식(6)와 같이 2차 베지어 곡선(quadratic Bezier curve) 형태로 전개하였다.

$$\boldsymbol{n_t = n_0 \cdot (1-t)^2 + (n_0 + n_1 - \delta)/2 \cdot 2t \cdot (1-t) + n_1 \cdot t^2} \tag{6}$$

식(6)의 제어점(control points)은 $\boldsymbol{n_0}$, $\boldsymbol{(n_0 + n_1 - \delta)/2}$, $\boldsymbol{n_1}$ 가 된다. 2009년 Tang[3]은 제어점의 볼록 껍질 특성을 이용하여 $\boldsymbol{n_t}$ 를 포함하는 법선 원뿔을 구성하였다(그림4-5 (b)).

본 논문에서는 식(6)에 De casteljau' s 알고리즘을 적용하여 $t = \frac{1}{2}$ 을 기준으로 베지어 곡선을 두 부분으로 나누고 그 때 각각의 제어점을 구한다. 따라서 앞부분 $t \in [0, \frac{1}{2}]$ 에서 $\boldsymbol{n_t}$ 의 제어점은 $\boldsymbol{n_0}$, $\frac{(3n_0+n_1-\delta)}{4}$, $\frac{(2n_0+2n_1-\delta)}{4}$ 이고, 뒷부분 $t \in [\frac{1}{2}, 1]$ 에서 $\boldsymbol{n_t}$ 의 제어점은 $\frac{(2n_0+2n_1-\delta)}{4}$, $\frac{(n_0+3n_1-\delta)}{4}$, $\boldsymbol{n_1}$ 이다. $t \in [0, \frac{1}{2}]$ 에서 $\boldsymbol{n_t}$ 와 $t \in [\frac{1}{2}, 1]$ 에서 $\boldsymbol{n_t}$ 는 각각 제어점의 볼록 껍질 특성을 만족시키기 때문에 전체적으로 $t \in [0,1]$에서 $\boldsymbol{n_t}$ 는 5개의 제어점 $\boldsymbol{n_0}$, $\frac{(3n_0+n_1-\delta)}{4}$, $\frac{(2n_0+2n_1-\delta)}{4}$, $\frac{(n_0+3n_1-\delta)}{4}$, $\boldsymbol{n_1}$ 을 포함시키는 법선 원뿔을 만들면 $t \in [0,1]$ 에서 $\boldsymbol{n_t}$ 가 그 때의 법선 원뿔로 포함된다고 말할 수 있다(그림4-5 (c)).

2009년 Tang[3]은 3개의 제어점($\boldsymbol{n_0}$, $\frac{(n_0+n_1-\delta)}{2}$, $\boldsymbol{n_1}$)을 이용해 법선 원뿔을 만들기 위해 1997년 Provot[2]이 제안한 병합 방법을 그대로 적용하였다. $\boldsymbol{n_0}$ 와 $\boldsymbol{n_1}$ 에 1997년 Provot[2]의 병합 방법을 이용해 하나의 법선 원뿔 $B_1 = [\overrightarrow{x_1}, \alpha_1]$을 만들고 다시 법선 원뿔 $B_1 = [\overrightarrow{x_1}, \alpha_1]$와 나머지 $\frac{(n_0+n_1-\delta)}{2}$에 1997년 Provot[2]의 병합 방법을 적용해 최종 그림 4-5 (b)와 같은 법선 원뿔 $B_{leaf} = [\overrightarrow{x_{leaf}}, \alpha_{leaf}]$ 을 구성했다. 앞서 말했듯이 1997년 Provot[2]의 병합 방법을 이용해 법선 원뿔을 구성하게 되면 실제 $\boldsymbol{n_t}$ ($t \in [0,1]$)가 존재하지 않는 부분도 법선 원뿔

안에 포함되어 비효율적인 법선 원뿔이 형성된다.

따라서 본 논문에서는 5개의 제어점($\boldsymbol{n_0}$, $\frac{(3\boldsymbol{n_0}+\boldsymbol{n_1}-\boldsymbol{\delta})}{4}$, $\frac{(2\boldsymbol{n_0}+2\boldsymbol{n_1}-\boldsymbol{\delta})}{4}$, $\frac{(\boldsymbol{n_0}+3\boldsymbol{n_1}-\boldsymbol{\delta})}{4}$, $\boldsymbol{n_1}$)을 포함시키는 법선 원뿔을 형성하여 더 효율적인 법선 원뿔을 만들었다(그림4-5(c)). 이 때 5개의 제어점을 포함하는 법선 원뿔을 형성하는 방법은 2005년 Barequet[9]이 제시한 알고리즘을 이용하였다.

본 논문에서 제시하는 방법은 시뮬레이션이 동적인 상황에서 많은 컬링 효과를 얻을 수 있다. $\boldsymbol{n_t}$ 자체를 여러 번 나눔으로써 더 세밀한 법선 원뿔을 만들 수 있으나 실험 결과 2005년 Barequet[9]의 알고리즘을 적용해 법선 원뿔을 구성하는 비용이 컬링으로 얻을 수 있는 이득 시간보다 커질 수 있으므로 5개까지만 제어점을 만들었다.

제 5 장 결 과

본 논문의 실험은 Intel Core i7-980 3.33GHz CPU을 이용하여 실행하였다.

연속 충돌 시뮬레이션에 본 논문의 방법을 적용해서 충돌 처리를 하는 모습을 보인다면 불연속 충돌 시뮬레이션에서도 당연히 논문의 방법이 유효할 것임으로 연속 충돌 시뮬레이션에서 본 논문의 방법들을 적용해 시뮬레이션한 결과만 제시하였다.

제 1 절 실제 시뮬레이션 결과 모습

만약 2009년 Tang[3]의 방법을 적용해 충돌 처리를 진행한 것과 본 논문에서 제시하는 방법을 적용해 충돌 처리를 진행한 것의 시뮬레이션 결과가 같다면 본 논문의 방법이 유효하다고 주장할 수 있다(그림 5-1).

(a) 시뮬레이션 진행 전 (b) 시뮬레이션 진행 후 (ground truth) (c) 시뮬레이션 진행 후 (본 논문의 방법)

그림 5-1. 원피스를 시뮬레이션 했을 때 2009년 Tang[3]의 방법을 통해 시뮬레이션한 결과와 본 논문의 방법을 통해 시뮬레이션한 결과가 같음을 볼 수 있다.

제 2 절 충돌 탐지 효율

본 논문에서 제시한 연속 충돌 법선 원뿔의 효율성을 보기 위해서 총 5개의 경우를 실험하였다. 원피스, 주름 치마, 블라우스, 코트, 흔들리는 천 이렇게 5개의 경우를 가지고 실험을 진행하였다. 2009년 Tang[3]의 방법을 기준으로 비교 실험을 진행하였고 각각의 방법을 통해 형성된 법선 원뿔을 이용해 충돌 검사를 진행해야 하는 삼각형 쌍의 개수를 줄인 결과를 제시하였다. 또한 이로 인한 시간적 성능 향상을 제시한다.

그림 5-3, 그림 5-5, 그림 5-7, 그림 5-9를 통해 처음 1~2frame은 40% 이상의 충돌 검사를 진행해야 하는 삼각형 쌍의 개수를 줄일 수 있는 것을 관찰할 수 있다. 반면 평균적인 충돌 검사를 진행해야 하는 삼각형 쌍의 개수는 30%정도만 향상되는 모습을 볼 수 있다. 이러한 결과를 보이는 이유는 옷을 구성하는 부분부터 시뮬레이션을 진행하였기 때문이다. 옷을 시뮬레이션 할 때 1~2 frame에서는 옷이 구성되면서 상당히 동적인 시뮬레이션이 된다.

본 논문에서 제시한 방법은 동적인 시뮬레이션일수록 큰 효율을 보이므로 그림 5-3, 그림 5-5, 그림 5-7, 그림 5-9과 같은 결과를 보인다고 말할 수 있다.

큰 천의 양쪽을 걸어두고 중력으로 인하여 흔들리게 만든 흔들리는 천 실험에서는 처음부터 구성되어 있는 천을 이용하여 동적 움직임을 만들어낸 실험이다. 또한 모든 삼각형들이 한 천으로 구성되어 있기 때문에 본 논문의 방법을 적용했을 때 큰 컬링 효과를 보이는 것을 확인할 수 있었다.

그림 5-2. 원피스 실험. 점 10,783개, 삼각형 20,849개, 선 31,633개로 이루어진 원피스를 시뮬레이션한 결과이다. Frame은 총 1,156번 진행되었다.

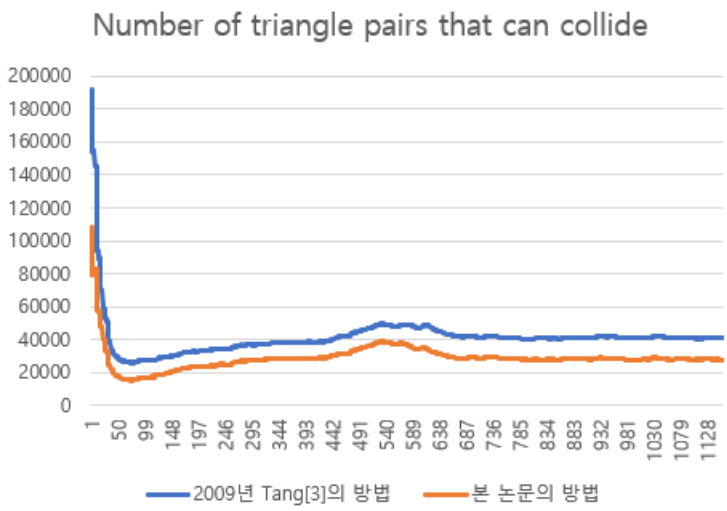


그림 5-3. 원피스 실험 결과. x축은 frame 번호를 의미하며, y축은 2009년 Tang[3]의 방법과 본 논문의 방법으로 구성된 법선 원뿔 트리를 이용해 컬링을 진행한 후에 충돌 검사를 실행해야 하는 삼각형 쌍의 개수를 의미한다.

원피스 실험	2009년 Tang[3]	본 논문의 방법
Number of triangle pairs(개)	41,213	28,832
Construction time(ms)	9.27	11.94
Traverse time(ms)	24.41	18.39
Total time(ms)	33.68	30.29

표 5-1. 원피스 실험 결과.
총 1,156 frame을 진행하면서 측정된 평균값들을 나타낸다.
순서대로 충돌 검사를 실행해야 하는 삼각형 쌍의 평균 개수,
법선 원뿔 트리의 평균 구성 시간,
법선 원뿔 트리 평균 탐색 시간,
구성 시간과 탐색 시간을 합한 최종 시간.

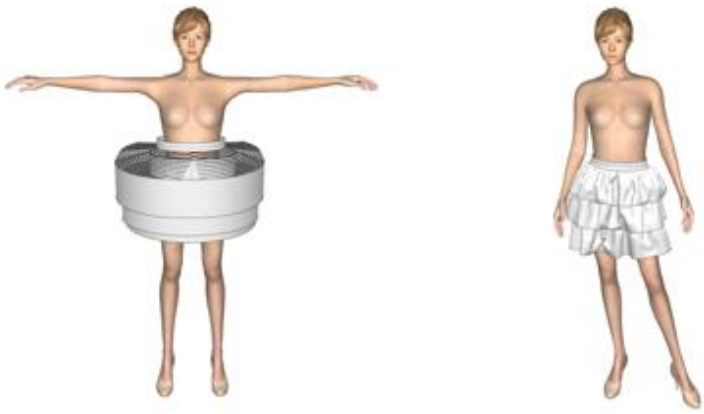


그림 5-4. 주름치마 실험. 점 14,313개, 삼각형 25,897개, 선 40,210개로 이루어진 주름치마를 시뮬레이션한 결과이다. Frame은 총 1,915번 진행되었다.

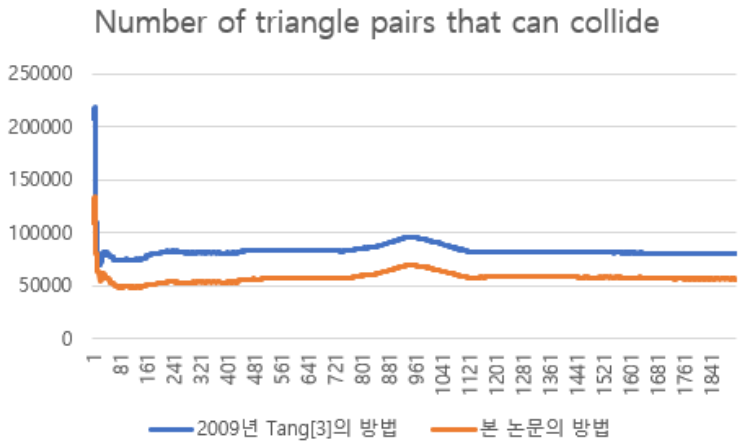


그림 5-5. 주름치마 실험 결과. x축은 frame 번호를 의미하며, y축은 2009년 Tang[3]의 방법과 본 논문의 방법으로 구성된 법선 원뿔 트리를 이용해 컬링을 진행한 후에 충돌 검사를 실행해야 하는 삼각형 쌍의 개수를 의미한다.

주름치마 실험	2009년 Tang[3]	본 논문의 방법
Number of triangle pairs(개)	83,543	58,137
Construction time(ms)	6.99	8.70
Traverse time(ms)	39.86	30.07
Total time(ms)	46.85	38.77

표 5-2. 주름치마 실험 결과.
총 1,915 frame을 진행하면서 측정된 평균값들을 나타낸다.
순서대로 충돌 검사를 실행해야 하는 삼각형 쌍의 평균 개수,
법선 원뿔 트리의 평균 구성 시간,
법선 원뿔 트리 평균 탐색 시간,
구성 시간과 탐색 시간을 합한 최종 시간.

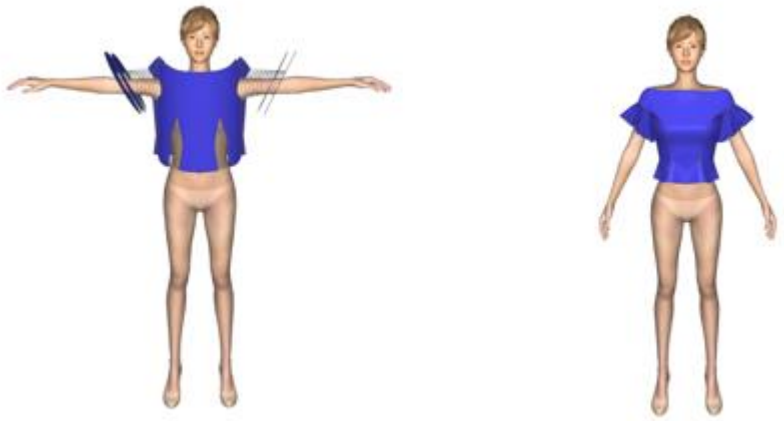


그림 5-6. 블라우스 실험. 점 55,11개, 삼각형 9,952개, 선 15,456개로 이루어진 블라우스를 시뮬레이션한 결과이다. Frame은 총 500번 진행되었다.

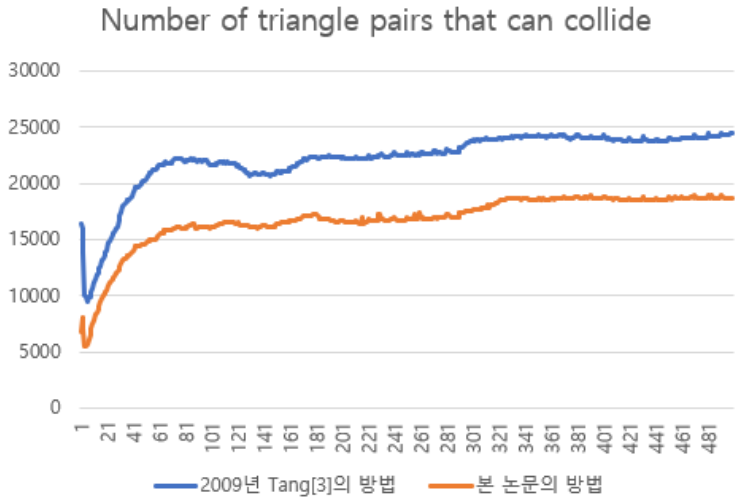


그림 5-7. 블라우스 실험 결과. x축은 frame 번호를 의미하며, y축은 2009년 Tang[3]의 방법과 본 논문의 방법으로 구성된 법선 원뿔 트리를 이용해 컬링을 진행한 후에 충돌 검사를 실행해야 하는 삼각형 쌍의 개수를 의미한다.

블라우스 실험	2009년 Tang[3]	본 논문의 방법
Number of triangle pairs(개)	22,173	16,782
Construction time(ms)	2.67	3.42
Traverse time(ms)	11.48	9.42
Total time(ms)	14.15	12.84

표 5-3. 블라우스 실험 결과.
총 500 frame을 진행하면서 측정된 평균값들을 나타낸다.
순서대로 충돌 검사를 실행해야 하는 삼각형 쌍의 평균 개수,
법선 원뿔 트리의 평균 구성 시간,
법선 원뿔 트리 평균 탐색 시간,
구성 시간과 탐색 시간을 합한 최종 시간.

그림 5-8. 코트 실험. 점 9,533개, 삼각형 18,713개, 선 28,257개로 이루어진 코트를 시뮬레이션한 결과이다. Frame은 총 250번 진행되었다.

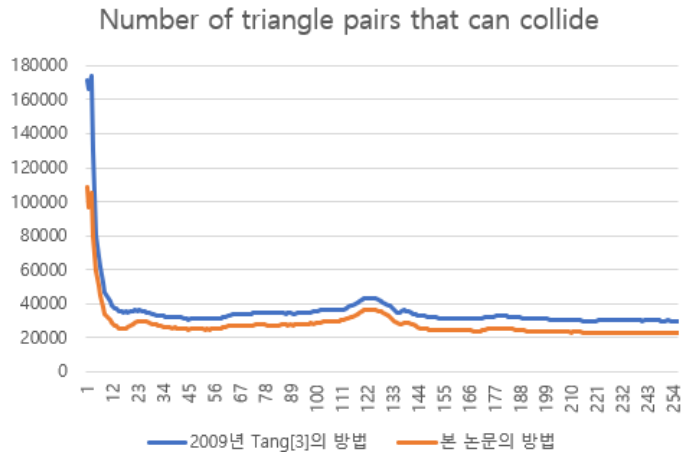


그림 5-9. 코트 실험 결과. x축은 frame 번호를 의미하며, y축은 2009년 Tang[3]의 방법과 본 논문의 방법으로 구성된 법선 원뿔 트리를 이용해 컬링을 진행한 후에 충돌 검사를 실행해야 하는 삼각형 쌍의 개수를 의미한다.

코트 실험	2009년 Tang[3]	본 논문의 방법
Number of triangle pairs(개)	35,938	27,862
Construction time(ms)	5.01	6.34
Traverse time(ms)	16.91	13.61
Total time(ms)	21.92	19.95

표 5-4. 코트 실험 결과.
총 250 frame을 진행하면서 측정된 평균값들을 나타낸다.
순서대로 충돌 검사를 실행해야 하는 삼각형 쌍의 평균 개수,
법선 원뿔 트리의 평균 구성 시간,
법선 원뿔 트리 평균 탐색 시간,
구성 시간과 탐색 시간을 합한 최종 시간.

그림 5-10. 천 실험. 천의 양쪽 끝점을 고정시킨 후 중력을 가해서 흔들리게 만든 실험이다. 점 14,657개, 삼각형 28,802개, 선 43,458개로 이루어진 천을 시뮬레이션한 결과이다. Frame은 총 400번 진행되었다.

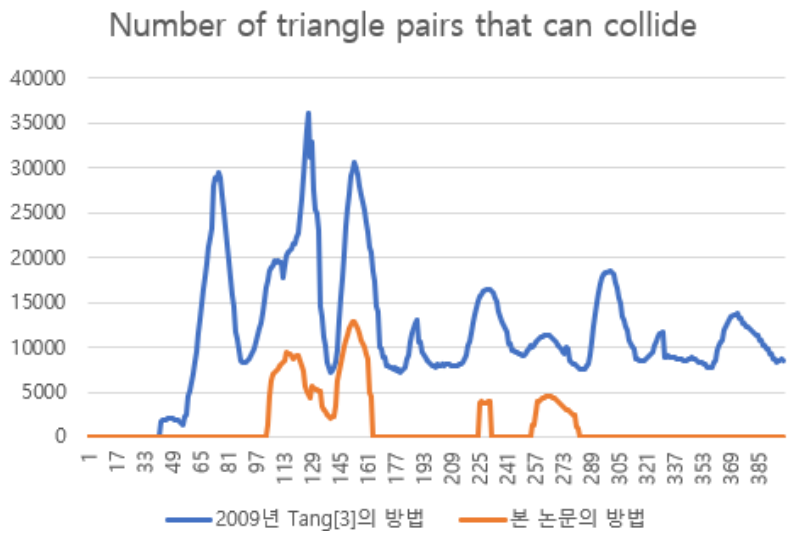


그림 5-11. 천 실험 결과. x x축은 frame 번호를 의미하며, y축은 2009년 Tang[3]의 방법과 본 논문의 방법으로 구성된 법선 원뿔 트리를 이용해 컬링을 진행한 후에 충돌 검사를 실행해야 하는 삼각형 쌍의 개수를 의미한다.

천 실험	2009년 Tang[3]	본 논문의 방법
Number of triangle pairs(개)	11,261	1,367
Construction time(ms)	7.72	10.30
Traverse time(ms)	6.72	2.28
Total time(ms)	14.44	12.58

표 5-5. 천 실험 결과.
총 400 frame을 진행하면서 측정된 평균값들을 나타낸다.
순서대로 충돌 검사를 실행해야 하는 삼각형 쌍의 평균 개수,
법선 원뿔 트리의 평균 구성 시간,
법선 원뿔 트리 평균 탐색 시간,
구성 시간과 탐색 시간을 합한 최종 시간.

제 6 장 결 론

제 1 절 연구 의의

옷과 같이 변형 가능한 모델(deformable model)을 시뮬레이션 함에 있어서 충돌 탐지는 굉장히 많은 비용을 요구한다. 강체를 시뮬레이션 하는 것과 달리 변형 가능한 모델을 시뮬레이션 할 때는 모든 점과 삼각형들이 어디로 운동할지 예측할 수 없기 때문이다. 이에 따라 현재까지 변형 가능한 모델을 시뮬레이션 함에 있어 시간을 줄이기 위한 많은 컬링 방법(culling method)에 관한 논문이 나와있는 상태이다. 그 중 법선 원뿔을 이용하는 컬링 방법은 1997년 Provot[2]에 의해 처음 등장한 이 후 2009년 Tang[3]에 의해 연속 충돌 탐지에 적용되었다. 그 후 많은 변형 가능한 모델 시뮬레이션에 이용이 되고 있다.

본 논문에서는 앞서 나온 법선 원뿔 컬링 방법을 좀 더 발전시켜 보다 빠른 충돌 탐지를 가능하게 하였다.

제 2 절 한계점 및 추후 연구

실험 결과, 법선 원뿔을 이용하여 많은 컬링 효율 증가를 이루어 냈다. 하지만 실제 시간을 측정시 컬링 효율에는 미치지 못 하는 결과를 보인다. 이는 좀 법선 원뿔을 구성할 때 더 많은 연산이 필요하기 때문이다. 법선 원뿔을 구성할 때 더 많은 연산이 필요한 이유는 크게 두가지이다. 첫 번째는 법선 원뿔 트리를 구성할 때 상향식(bottom-up)으로 법선 원뿔 노드들을 구성한다. 이 때 두 개의 자식 법선 원뿔을 병합하여 부모 법선 원뿔을 구성하는데, 병합 방법 자체의 연산이 많아졌기 때문이다. 두 번째는 잎 노드(leaf node) 법선 원뿔을 구성할 때 2005년 Barequet[9]이 제시한 알고리즘의 연산이 복잡하기 때문이다. 실험 결과 첫 번째 이유인 병합 방법 자체의 연산양은 큰 차이를 보이지 않았다. 하지만 두 번째 이유인 잎 노드 구성시에는 많은 연산양의 차이를 보였다.

옷이 만들어진 이후에는 비교적 정적인 시뮬레이션이 진행된다. 따라서 본 논문에서 제시하는 동적인 상황을 고려하여 고안된 잎 노드 구성법은 오히려 정적인 상황에서 간접비(overhead cost)로 작용한다. 따라서, 추후에는 동적인 상황과 정적인 시뮬레이션 상황을 고려하여

선택적으로 잎 노드 법선 원뿔을 구성하는 방법에 관해 연구할 계획이다.

참고 문헌

[1] P. VOLINO and N. M. THALMANN, "Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity," *Computer Graphics Forum*, vol. 13, no. 3, pp. 155-166, Aug. 1994.

[2] X. Provot, "Collision and self-collision handling in cloth model dedicated to design garments," in *Eurographics*, Springer Vienna, 1997, pp 177-189.

[3] Min Tang, S. Curtis, Sung-Eui Yoon, and D. Manocha, "ICCD: Interactive Continuous Collision Detection between Deformable Models Using Connectivity-Based Culling," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 4, pp. 544-557, Jul. 2009.

[4] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, 1998.

[5] K.-J. Choi and H.-S. Ko, "Stable but responsive cloth," in *ACM SIGGRAPH 2005 Courses on - SIGGRAPH '05*, 2005.

[6] M. Tang, R. Tong, R. Narain, C. Meng, and D. Manocha, "A GPU-based Streaming Algorithm for High-Resolution Cloth Simulation," *Computer Graphics Forum*, vol. 32, no. 7, pp. 21-30, Oct. 2013.

[7] R. Tamstorf, T. Jones, and S. F. McCormick, "Smoothed aggregation multigrid for cloth simulation," *ACM Transactions on Graphics*, vol. 34, no. 6, pp. 1-13, Nov. 2015.

[8] M. Tang, D. Manocha, J. Lin, and R. Tong, "Collision-streams,"

in *Symposium on Interactive 3D Graphics and Games on - I3D ' 11*, 2011.

[9] G. Barequet and G. Elber, "Optimal bounding cones of vectors in three dimensions," *Information Processing Letters*, vol. 93, no. 2, pp. 83-89, Jan. 2005.

Abstract

Efficient normal cone culling method for collision detection of deformable model

Chang-Jin Lee
Department of Electrical and Computer Engineering
The Graduate School
Seoul National University

Unlike rigid body simulation, the simulation of deformable model (fabric simulation) require collision detection of all triangle pairs within the same object, which takes a great deal of time. For this reason, it is difficult to create realistic physical-based deformable mode simulation in real time using current graphic technology and hardware.

Accordingly, many acceleration studies have been conducted to reduce collision detection time in deformable model simulation. The method proposed in this paper was inspired by a study to reduce the collision detection operation using the surface normal vector of the fabric.

In this paper, we develop an existing method using a surface normal vector and propose a method to efficiently reduce collision detection time using a surface normal vector. The method presented in this paper can be applied to both discrete collision detection and continuous collision detection. When applied to continuous collision detection, the method presented in this paper shows good performance, especially in dynamic scenes.

As a result, the method introduced in the paper showed a 10-40% reduction in the number of triangle pairs that need to be tested for collisions compared to the existing method. In the method of this paper, trade-off occurs at the time of constructing the binary tree, but there is an improvement in performance as a whole.

Keywords : Normal cone, Collision detection, Cloth simulation, Continuous collision detection, Discrete collision detection
Student Number : 2018−26787