알란 웨이크(*Alan Wake*)에서 본 클로스 시뮬레이션(cloth simulation)의 비밀

The Secrets Of Cloth Simulation In *Alan Wake*

헨리크 엥퀴비스트(Henrik Enqvist)

2010년 8월 29일

http://www.gamasutra.com/view/feature/4383/the_secrets_of_cloth_simulation_in_.php

[애니메이션 프로그래머인 헨리크 엥퀴비스트는 마이크로소프트사가 출시한 엑스박스 360 의 호러 스릴러물 알란 웨이크(Alan Wake)에서 주요 캐릭터의 재킷에서 보여지는 실감나는 트위드(tweed) 클로스 시뮬레이션(cloth simulation)을 그의 팀이 어떻게 생성했는지 보고자 한다.]

액션 스릴러의 주요 캐릭터는 '알란 웨이크'로 행방불명 된 그의 아내에 대한 미스터리를 풀려고 하는 동시에 어둠의 세력과 싸워야 하는 악몽 같은 시나리오에 사로잡혀 있는 작가이다. 그는 잘 훈련된 액션 히어로라기 보다는 일상적으로 볼 수 있는 평범한 사람이다.

그의 캐릭터를 설정하기 위해 아트 디렉터는 그가 무장을 한 채 낡은 트위드 재킷을 입었으면 했다. 이 게임은 실제 세상을 설정해 놓은 곳에서 벌어지며, 판타지 게임이나 스페이스 슈터와 비교해 보면 캐릭터의 개성을 만들어 주는 툴은 한정적이다. 그렇기 때문에 캐릭터가 입고 있는 의상이 훨씬 더 중요해진다.

알란 웨이크의 재킷은 게임 플레이어에게 스릴러의 분위기가 나도록 최대한 현실성이 있어야 한다. 재킷은 바람에 날려야 하고 캐릭터가 숲으로 돌진할 때는 캐릭터에 따라 추가적인 움직임이 있어야 한다. 프로그래머로서 당신은 클로스 시뮬레이션에 대해 즉시 생각해 보게 된다. 수 많은 게임에 이미 클로스 시뮬레이션이 적용되어 있다. 하지만, 이러한 방법들은 종종 실크나 고무의 느낌은 준다. 어떤 것은 마음에 들지 않았다. 최근에서야 클로스 시뮬레이션에 대해 훌륭한 타사 솔루션이 일부 적용되고 있지만, 안정적인 솔루션을 필요로 할 때면 그러한 툴은 존재하지 않았거나 우리의 필요를 충족시키지 못했다.

이 글은 우리가 마주치는 문제점을 진단해 보고 자체적인 클로스 시뮬레이션을 구동하기 위한 솔루션을 제시한다. 여기에 기술한 방법을 잘 안다고 하더라도 여러 요소를 모으는 방법은 본 기사를 읽는데 흥미로움을 가져다 줄 것이다.

재킷 복장

재킷은 일반적인 스킨드 메쉬(skinned mesh)로서 캐릭터의 나머지 부분과 함께 설계된다. 재킷 메쉬를 구동하는 골격은 일반적인 골격의 위에 있는 개별적인 한 층이다. 재킷의 소매는 일반적으로 윗팔과 아래팔로 구성된다. 이러한 윗팔과 아래팔은 하나의 주요 골격과 하나의 구부러진 골격으로 나누어 진다. 상부 재킷은 룩엣 콘스트레인트(look-at constraints)로 구동되고 Verlet 시뮬레이션은 하부 재킷을 구동한다.

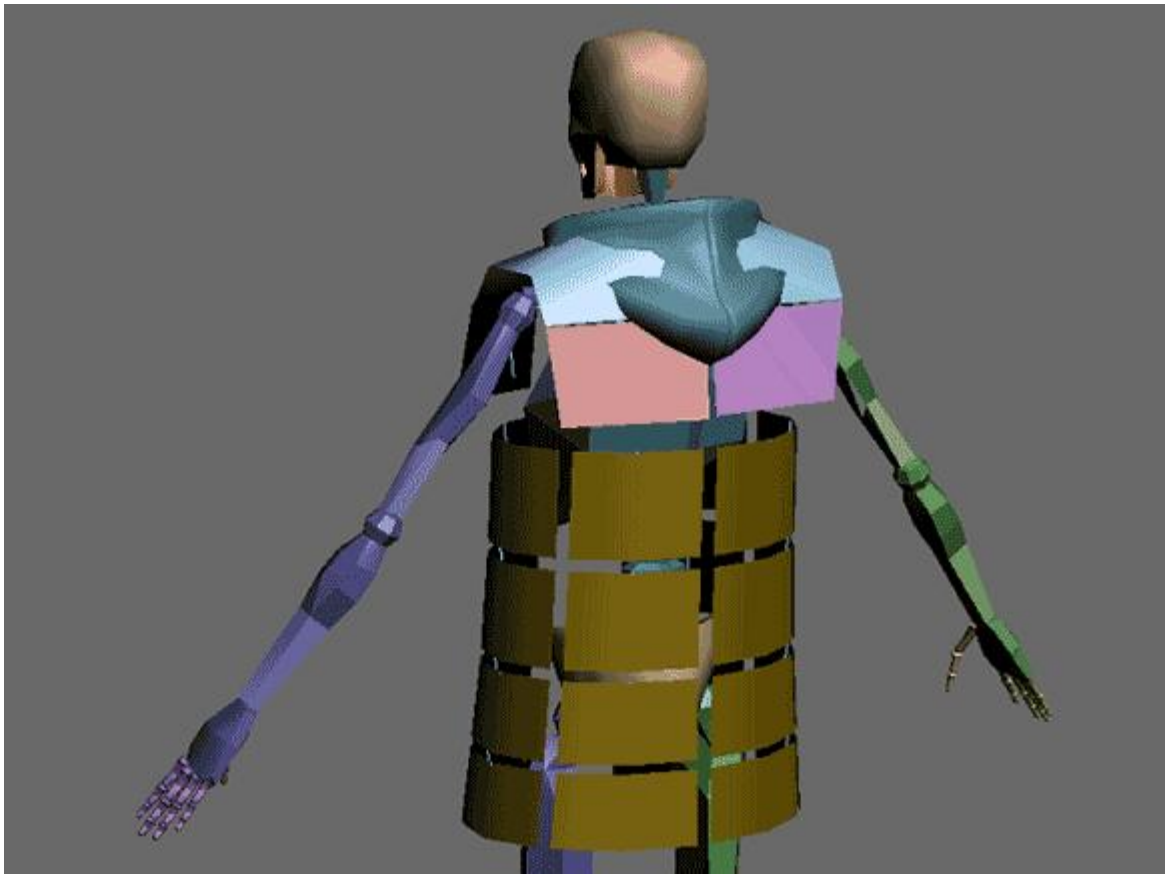

그림 1. 일반적인 게임 골격의 상부에 있는 재킷 복장

상부 재킷

상부 재킷 골격은 탑-다운 방식으로 상부의 뼈가 움직이면 하부의 뼈도 따라서 움직인다. 하부 뼈가 흉부로 곧장 움직일 수 있도록 했다. 그랬더니 캐릭터의 움직임이 둔해졌다. 특히, 캐릭터가 그의 어깨를 올릴 때 재킷이 수직으로 잘 움직여지지 않았다.

상부 재킷에서 윗팔의 부분들 쪽으로 어깨뼈를 룩엣 콘스트레인트로 구동해 실제 어깨 패드의 움직임을 재현했다. 어깨 패드가 윗팔의 움직임을 따르고, 팔을 올릴 때 어깨 패드가 나머지 뼈들을 들어 올린다. 이는 마치 실제 재킷과 같다.

이러한 일련의 상황에서 다음과 같은 골격은 상부 재킷 및 시뮬레이션된 하부 재킷 간의 한 층이다. 이러한 골격은 룩엣 콘스트레인트를 통해 바로 아래로 구동되기 때문에 어깨로 인한 회전을 보상한다. 또한, 좌측 뼈와 우측 뼈간 포지션 콘스트레인트(position constraints)를 추가해서 어깨 패드가 움직일 때 발생하는 스트레칭을 보상한다.

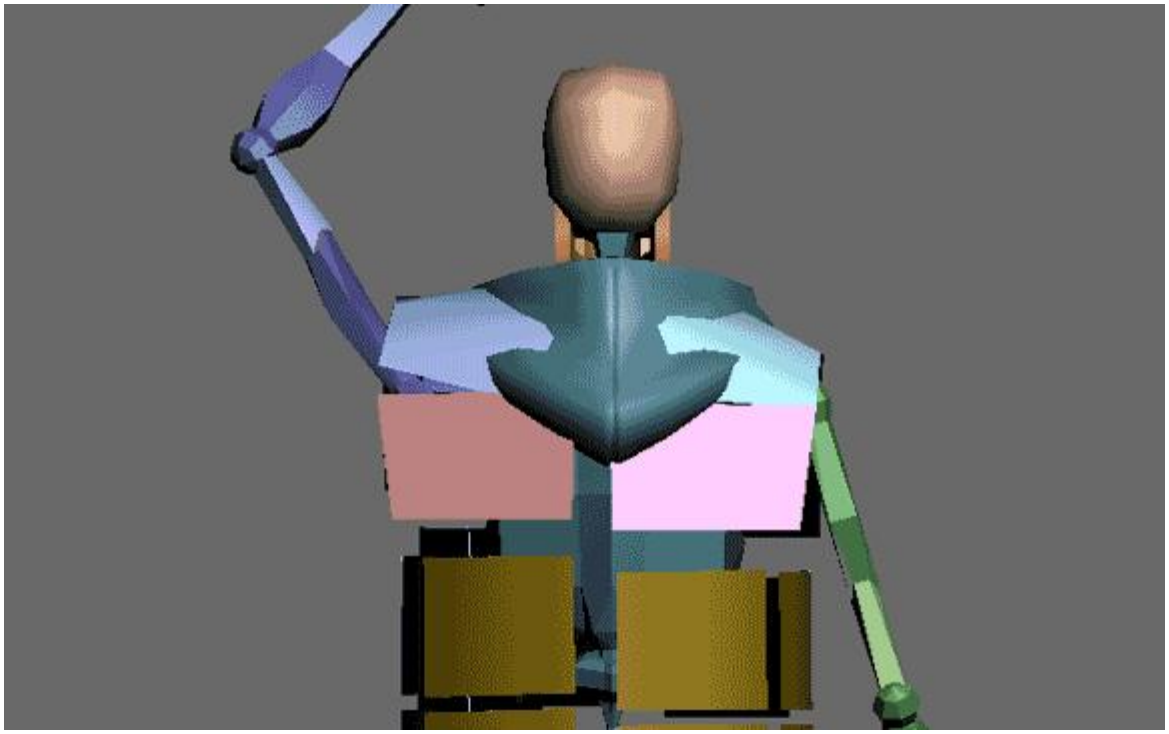

그림 2. 캐릭터가 팔을 올릴 때 골격의 움직임

애니메이션 익스포터(animation exporter)에서 콘스트레인트을 구현하고 이 결과를 애니메이션 데이터에 제공하는 훌륭한 솔루션일 수 있었다. 하지만, 게임 엔진 내에서 실시간 골격을 구동할 수 있기를 원했다.

이 방법은 애니메이션 데이터에서 바이트 일부를 줄일 수 있고 캐릭터가 재킷을 입었는지의 여부와 상관없이 캐릭터간 애니메이션을 쉽게 공유할 수 있었다. 또한, 인게임 IK 로 인해 발생하는 어깨의 움직임은 콘스트레인트를 실시간으로 해결할 때 정확하게 적용하게 된다.

하부 재킷

상부 재킷을 해결한 후 재킷의 하부를 시뮬레이션 하는 것을 보자. 게임에서 대부분의 클로스 시뮬레이션은 클로스 시뮬레이션에서의 정점(vertex)들과 렌더링된 메쉬의 정점들간 1 대 1 매핑을 가진다.

우리의 경우, 재킷 메쉬의 정확성을 보존하고 코더 딕테이티드 콘스트레인트(coder-dictated constraint)에 의한 방해를 받지 않기를 원했다. 예를 들여, 렌더링 측면에서 시뮬레이션을 위해 동일한 메쉬를 받아들였어야 했으면 재킷의 주머니와 전면의 실루엣은 손상되었을 것이다.

일반적인 맵은 재킷의 형상을 표현하는데 유용했지만 충분하지 않은 것 같다. 대신, 우리는 아티스트들이 자유롭게 원하는 모습으로 재킷의 형상을 만들기 원하여 일반적인 맵을 사용해서 입체 형상을 줄이기 보다는 주름이나 다른 상세 사항을 추가하려고 했다.

이에 대한 우리의 솔루션은 재킷을 시뮬레이션하는데 사용한 저해상도의 클로스 메쉬를 가진 다음, 스킨드 메쉬를 구동하는데 사용했던 골격에서 이 클로스 메쉬를 매핑하는 것이었다.

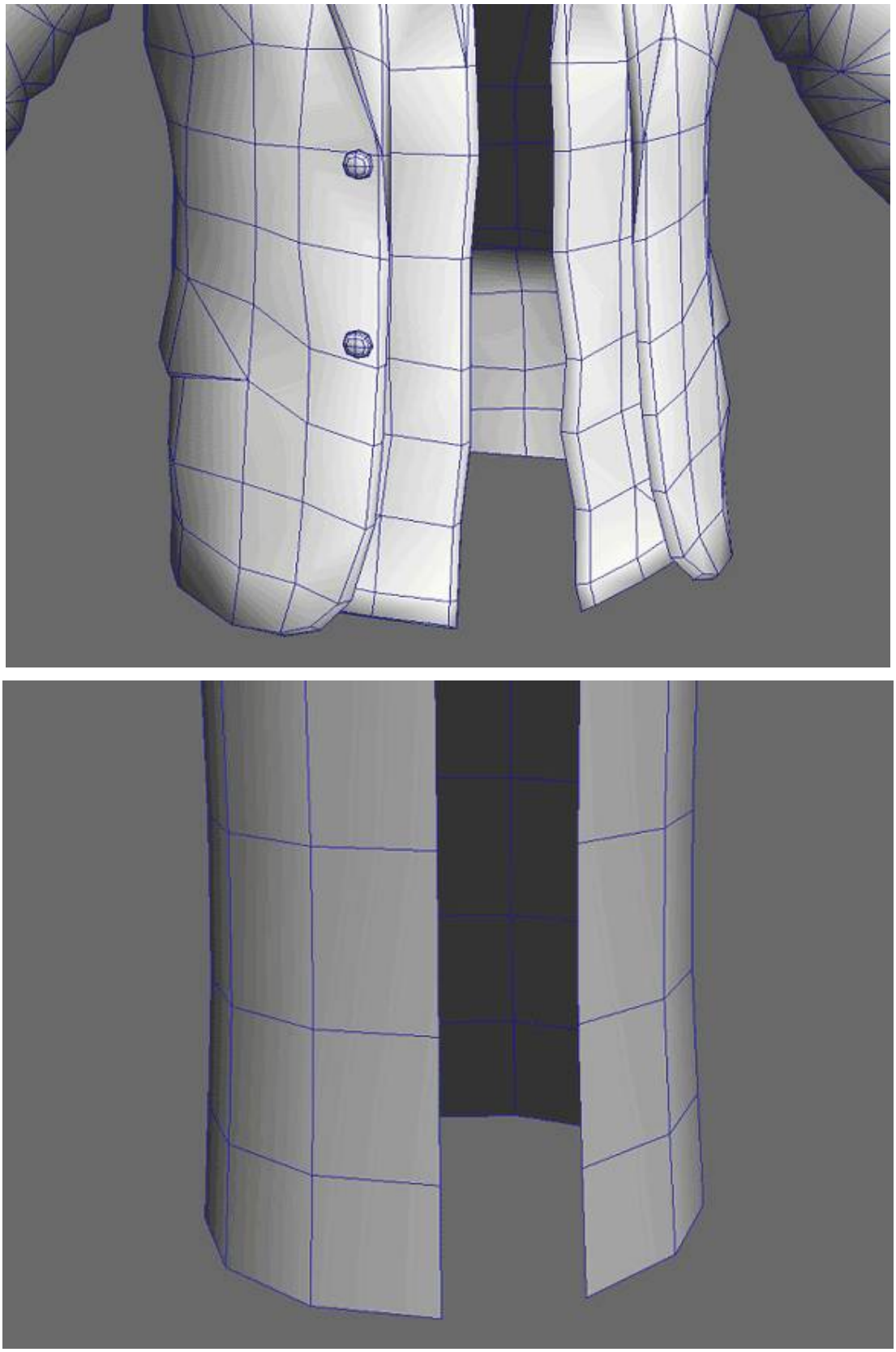

그림 3. 재킷의 실루엣 vs. 시뮬레이션을 통한 정점을 공유하는 재킷

Verlet 물리학

우선, 우리는 Verlet 물리학을 살펴본 후 이 시뮬레이션이 골격에 어떻게 매핑되는지 보자. Verlet 물리학은 현재 게임에서 클로스 시뮬레이션을 해결하기 위한 표준이다. Verlet 에 대해 잘 알지 못한다면 다음과 같은 가마수트라(Gamasutra)의 글(Devil in the Blue Faceted Dress: Real Time 클로스 Animation 또는 Advanced Character Physics)을 읽어보기 바란다.

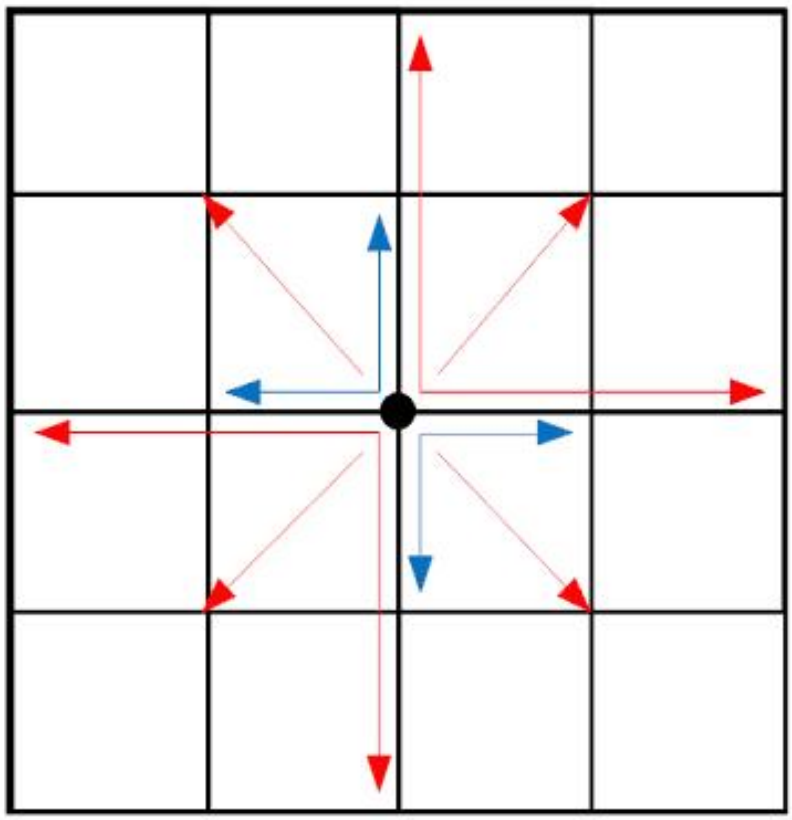

그림 4. 정점의 4x4 그리드와 정점 중 하나에 대한 콘스트레인트

이것이 어떻게 작용하는지 간단히 설명하자면, 그림 4 는 클로스 메쉬와 이러한 정점의 하나를 위한 스프링 콘스트레인트(spring constraints)를 나타낸다. 그림에서 나타난 것과 같이 메쉬의 각 정점은 인근 모든 정점과 연결된다.

즉각적인 네이버에 대한 콘스트레인트는 스트래치 콘스트레인트(stretch constraint)라 불리고 파란색으로 나타난다. 붉은색으로 나타난 긴 콘스트레인트는 쉬어/벤드 콘스트레인트(shear/bend constraint)라 불린다.

향후 상이한 파라미터와 함께 이들을 해결할 것이기 때문에 이러한 콘스트레인트를 두 그룹으로 나누는 것이 중요하다. 재킷에서 클로스의 상부 열은 캐릭터에 덮여서 시뮬레이션으로 구동되지 않는다.

그리드 같은 메쉬를 가지는 것은 알고리즘 자체로 요건은 아니지만 클로스 시뮬레이션의 측면에서 이 토폴로지는 매우 용이하게 작업을 할 수 있도록 한다. 클로스 시뮬레이션의 중심은 두 부분으로 구성된다. 첫 번째 부분은 Verlet 통합으로 각 정점 별로 속도를 계산하고 이를 위치에 적용한다.

```
Vector3 vVelocity = vertex.vCurrentPosition - vertex.vPreviousPosition;
vertex.vPreviousPosition = vertex.vCurrentPosition;
vertex.vCurrentPosition += vVelocity * ( 1.0f - fDampingFactor ) + vAcceleration * fDeltaTime *
fDeltaTime;
```

우리의 프로젝트에서 vAcceleration 는 중력과 바람의 합으로 주어진다. 댐핑은 재킷의 모양새를 변경하고 시뮬레이션을 안정적으로 만든다. 높은 댐핑 팩터(damping factor)는 재킷이 매우 가벼운 직물의 느낌을 주어서 느리고 부드럽게 움직이도록 하고 낮은 댐핑 팩터는 재킷을 무겁게 하여 움직임 뒤 재킷이 흔들리거나 떨리는 것을 더 오래가게 한다.
알고리즘의 두 번째 부분은 릴렉세이션(relaxation)으로 알려진 스프링 콘스트레인트를 해결하는 것과 관련이 있다. 각 콘스트레인트에 대해 서로의 정점으로 이동하거나 서로의 정점에서 벗어나서 움직이기 때문에 이러한 정점은 그들의 원래 길이를 만족시킨다. 읽을 수 있는 형태의 코드 스니펫(code snippet)은 다음과 같다.

```
Vector3 vDelta = constraint.m_vertex1.m_vCurPos - constraint.m_vertex0.m_vCurPos;
float fLength = vDelta.length();
vDelta.normalize();
Vector3 vOffset = vDelta * ( fLength - constraint.m_fRestLength );
constraint.m_vertex0.m_vCurrentPosition += vOffset / 2.0f;
constraint.m_vertex1.m_vCurrentPosition -= vOffset / 2.0f;
```

스트래치 콘스트레인트는 클로스와 함께 하도록 하는 반면에 쉬어/벤드 콘스트레인트는 클로스의 형상을 유지하도록 돕는다. 보다시피 이 시스템은 엄격하게 움직이는 클로스를 만든다. 그래서 우리는 새로운 위치를 해결하기 전에 계수를 쉬어/벤드 콘스트레인트에 추가할 것이다.

```
vOffset *= fStiffness;
constraint.m_vertex0.m_vCurrentPosition += vOffset / 2.0f;
constraint.m_vertex1.m_vCurrentPosition -= vOffset / 2.0f;
```

강도 계수가 1.0 이면 단단한 클로스를 만들지만, 0.0 이면 제한없이 클로스가 구부러질 수 있다.

고정된 타임 스텝

Verlet 통합은 이전의 타임 스텝이 현재의 타임 스텝과 동일하고, 그렇지 않다면 산출된 속도가 정확하지 않다고 가정한다는 것을 알고 있을 것이다. Verlet 통합을 사용할 때 변화 가능한 타임 스텝을 잘 사용할 수 있지만 해결하는 콘스트레인트는 타임 스텝에서 변경에 대해 용서하지 않는다.

해결자가 콘스트레인트를 반복해서 수행하기 때문에 콘스트레인트는 결코 완벽하게 해결되지 않을 것이라고 예상한다. 게임에서 이러한 부정확성은 스트레칭으로서 명백하게 되고, 당신의 타임 스텝이 짧을수록 당신이 보게 될 스트레칭도 적어진다.

결국, 이는 얼마나 많은 CPU 시간 동안 클로스에 시간을 투자할 것인지에 대한 균형이 된다. 타임 스텝이 일정하지 않아서 클로스의 스트레칭이 변경가능하고 원하지 않은 진동을 시스템에 적용하게 된다. 더 중요한 것은 타임 스텝은 강도 계수와 기타 클로스 파라미터에 영향을 미치고 더 짧은 타임 스텝은 동일한 강도 계수를 사용한다고 해도 본래 더 뻣뻣한 클로스를 제공한다.

실제로, 이는 당신의 클로스 파라미터를 사용해서 외형을 비트는 것을 시작하기 전에 정해진 타임 스텝 동안 자리를 잡아야 한다. 물리학에 대한 변경 가능한 스텝 타임이 있는 게임이 있는 것으로 알고 있다. 하지만, 내 경험상 물리학에 대한 타임 스텝과 게임 로직이 고정되어 있으면 당신의 삶은 훨씬 용이해 질 것이다.

후드

클로스 시뮬레이션에 대해 상세히 알아보도록 하기 전에 후드가 어떻게 시뮬레이션되는지 대략적으로 먼저 살펴보자. 후드 메시의 정점을 스키닝하기 위해 하나의 추가적인 뼈를 사용했다. 뼈의 중심에서 후드 뒤까지 추를 생성했다. 추의 끝은 Verlet 물리학으로 구동되는 단일의 작은 입자이다. 그러면, 이 뼈는 추로의 룩엣 콘스트레인트를 통해 타깃이 된다.

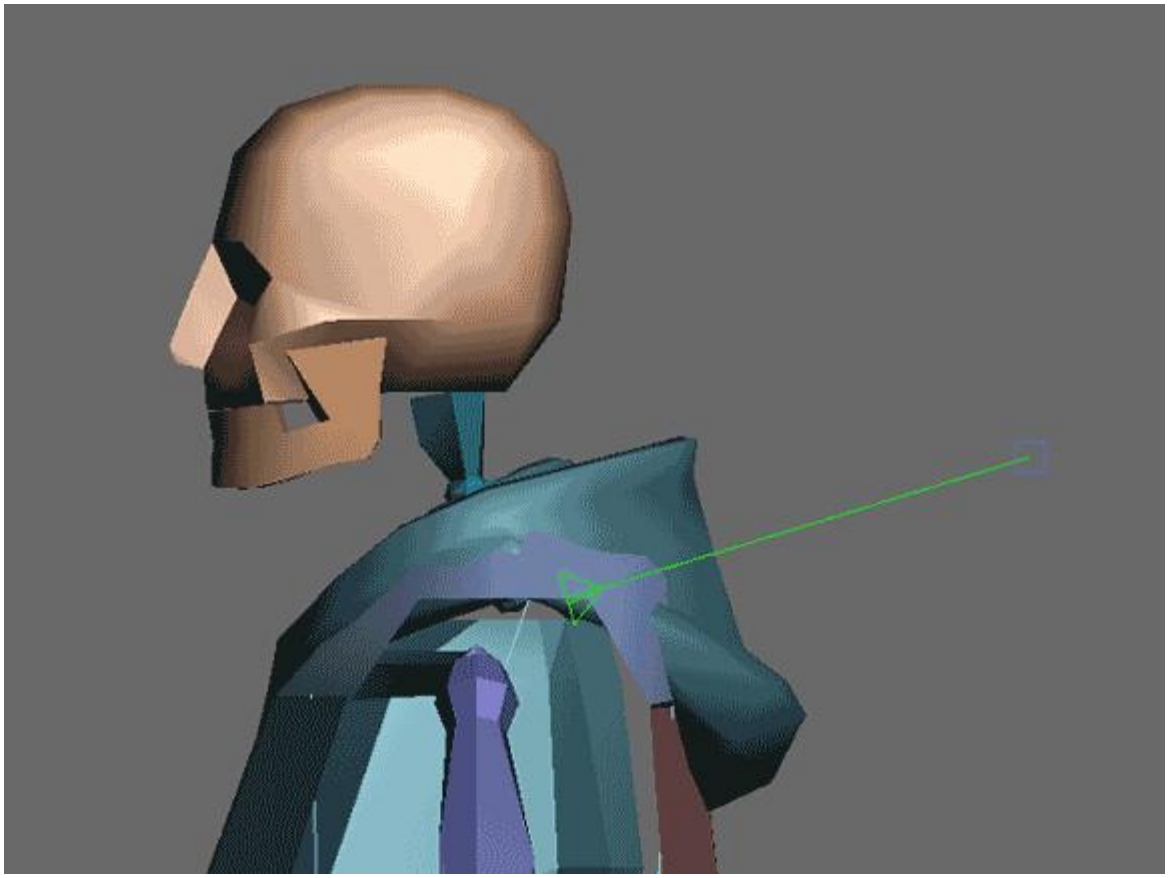

그림 5. 후드와 추

골격 매트릭스 생성

후드를 보면 하부 재킷을 위해 우리가 다음에 무엇을 해야 하는지 알 수 있다. 시뮬레이션된 메쉬에서 정점의 위치를 이용해서 골격의 변형을 계산하게 된다.

우리가 하는 첫 번째 일은 골격을 잘 조화해서 각 뼈의 피봇이 시뮬레이션된 메쉬의 정점과 맞는 것이다. 이는 매트릭스의 변형 부분을 설명하기 위한 사소한 것이다.

다음으로, 우리는 3x3 회전 매트릭스를 계산할 필요가 있다. 매트릭스의 각 열(또는 매트릭스 설정에 따라 행)은 뼈의 x-, y-, z-축으로 주어진다.

뼈의 x-축은 기본 정점에서 다음 정점으로의 방향을 정의한다. y-축은 좌측 정점에서 우측 정점으로 벡터로 주어진다.

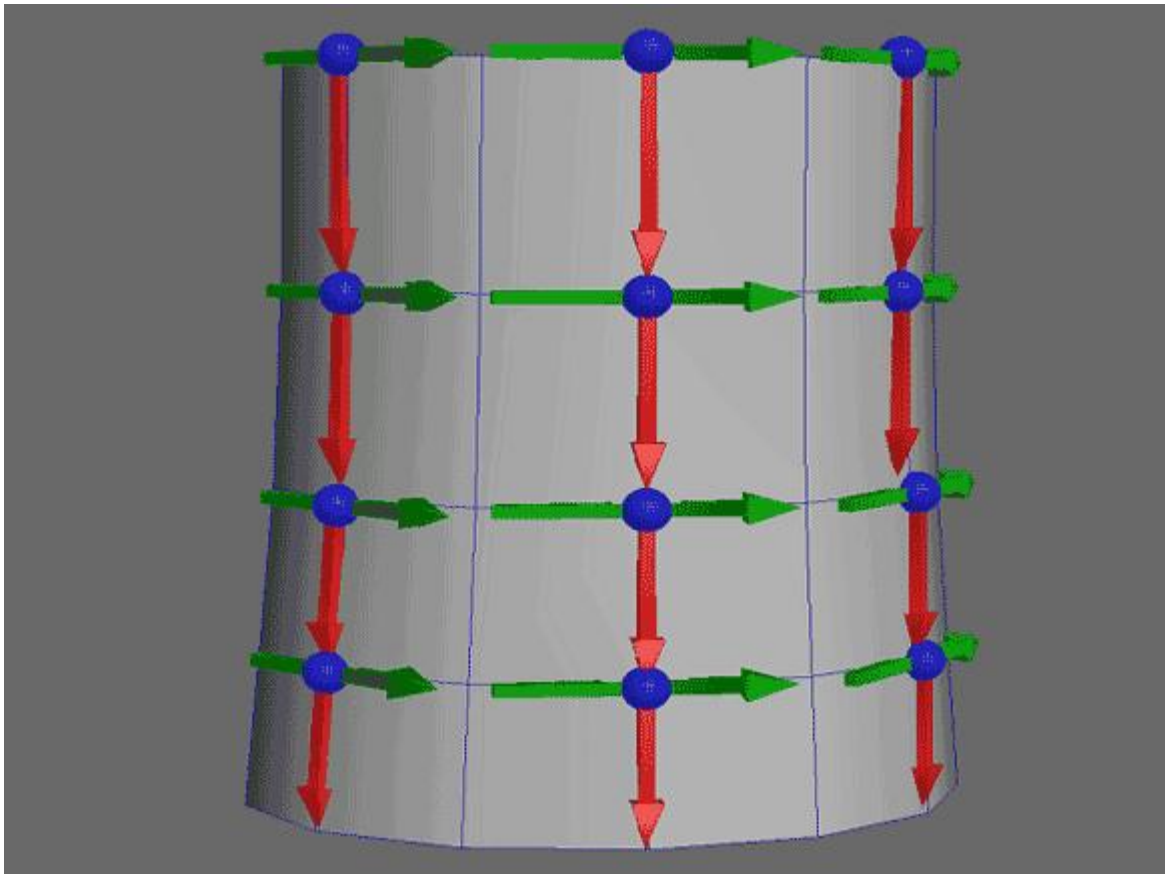

그림 6. 클로스 메쉬로 붙은 골격

그림 6 에서 x-축은 붉은 색으로 나타났고, y-축은 초록으로 나타났다. z-축은 이러한 축 간의 교차 제품으로 계산된다. 최후에는, 매트릭스를 직교-정규화(ortho-normalize)해서 변형에서 구부러짐을 제거한다.

보다시피 수직 방향에서 우리는 골격 설정을 위해 클로스 메쉬의 각 열(마지막 제외)을 사용하지만, 수형 방향에서는 오직 매 두 번째 행만 사용한다. 또한, 상기에 언급한 예술적인 이득 외에도 이는 매우 빠른 방법이다. 기존의 스키닝 기법은 GPU 측면에서 사용해 메쉬를 표현할 수 있었지만 대안은 큰 역동적인 정점 버퍼를 업데이트 했을 것이다.

클로스 메쉬는 상당히 낮은 해상도를 가질 수 있지만 CPU 측면에서는 더 용이하다. 우리의 솔루션에서 유일한 오버헤드는 낮은 해상도 시뮬레이션에서 높은 해상도 메쉬로의 전환이지만 정해진 오버헤드로는 시뮬레이션의 나머지와 비교해 무시할만하다.

충돌

다리와 바디에 대해 클로스의 클리핑을 해결하기 위해 우리는 타원체 vs. 입자 충돌 감지를 사용한다. 그림 7 은 캐릭터에 대한 재킷 클리핑을 해결하기 위해 필요한 타원체를 나타낸다.

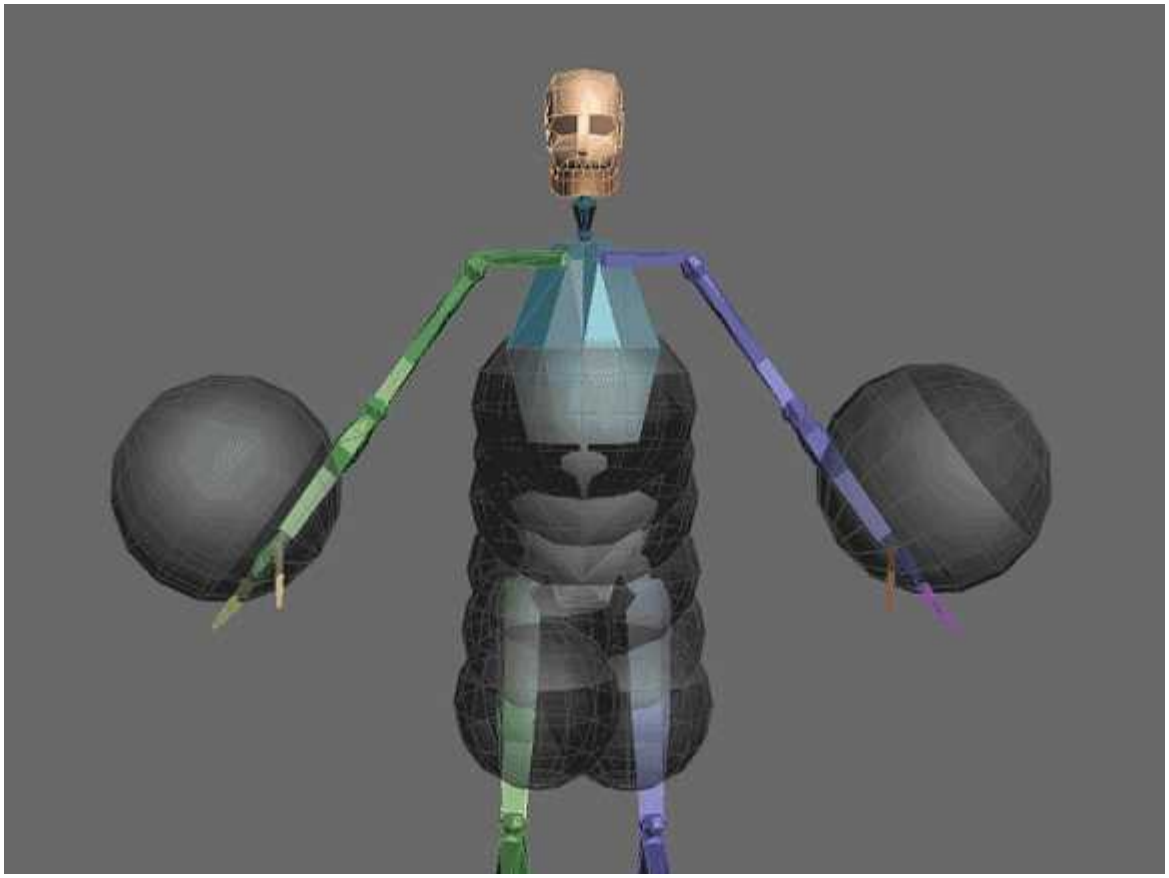

그림 7. 웨이크에 대한 타원체 구성

타원체는 입자에 대한 충돌 감지를 위해 매우 빠르다. 충돌은 타원체와 입자가 있는 공간을 변형해 해결할 수 있다. 그렇기 때문에, 타원체는 점차 구체가 된다. 그러면, 우리는 빠른 구체 vs. 입자 충돌 테스트를 적용할 수 있다.

특별히, 이는 타원체 길이, 폭, 높이 값을 사용하고 이 값을 입자의 위치에 적용해서 역변형을 생성하여 이행한다. 이와 관련해 유일한 문제는 우리의 좌표 시스템으로 변형한 후 얻게 되는 충돌은 구부러진다.

충돌의 방향을 해결할 때 우리는 약간의 비정확성 속에 살 수 있다고 결정했다. 이 경우, 심각하게 스트래치된 타원체는 매우 나쁜 응답을 야기할 수 있었을 것이고 우리는 이를 두 개 이상의 균일한 항목으로 나눈다.

최대 입자 거리

이를 해결하는데 또 다른 문제는 재킷의 안정성이다. 빠른 움직임에서의 클로스는 매듭(knot)을 야기할 수 있거나 충돌 볼륨이 잘못되어 끝난다. 우리는 시뮬레이션된 클로스에서 각 정점 별로 안전한 거리를 정의해서 이를 해결한다.

각 정점 별로 원래의 나머지 위치는 가장 가까운 뼈에 붙어있고 우리는 기준점으로 이를 사용한다. 시뮬레이션이 임계치를 초과하면 우리는 단순히 정점을 기준점으로 더 가까이 움직인다. 우리의 설정에서 아래의 정점이 어깨에 더 가까운 정점보다 더 많이 움직이도록 한다. 매듭과 클리핑의 드문 경우가 발생하기 전에 우리의 정점을 움직이도록 할 수 있는 최대 거리는 약 40cm 였다. 우리는 또한 충돌 평면(collision plane) 등의 다른 기법을 시도했지만 최대 거리 방법은 단연 최고의 방법이었다. 이는 설정하기에 빠르고 단순하며 클로스에서 시각적인 문제를 야기하기 전에 대부분의 움직임을 가능하게 한다.

트위드에 가깝고 고무에 먼

지금까지 우리는 우리의 목표를 달성하기 위한 훌륭한 방법을 사용했다. 우리의 아티스트들은 재킷을 모형화해서 만족시켰다. 모든 것이 게임에서 시뮬레이션되는 것 같이 애니메이터는 재킷을 애니메이션으로 만들 필요가 없고 CPU 는 다른 게임 관련 것을 위해 프로세싱 파워를 사용할 수 있기 때문에 좋다. 하지만 우리를 방해하는 한 가지가 있었으니 이것이 고무와 같다는 것이다.

스트레칭과의 전쟁

먼저, 스트레칭을 없애야 한다. 이전에 말했다시피 스트레칭 현상은 반복적인 알고리즘의 특성으로 야기된 에러로 생긴다. 이는 많은 사람들이 연구하기 원하는 주제이고, 이를 해결하기 위한 다양한 방법이 있음을 알게 될 것이다.

불행하게도 가능한 모든 솔루션은 우리의 소중한 CPU 의 전부를 클로스에 할당하도록 요구했다. 대신에, 우리는 마지막 스텝을 소위 “하드 콘센트레인트(hard constraint)”라 불리는 것을 도입한 클로스 시뮬레이션에 추가하여 스트래칭을 해결했다.

우리는 하드 콘센트레이트를 수직인 모든 스트래치 콘센트레이트가 되는 것으로 정의했다. 우리는 이러한 콘센트레이트를 탑다운으로 분류해서 어깨 근처의 콘센트레이트가 다리로 콘센트레이트가 내려가기 전에 해결한다.

이 콘센트레이트를 정확한 순서로 반복하기 때문에 쌍의 상부 정점이 이미 해결되었고 어떠한 스트레칭을 야기하지 않는다는 것을 알고 있기 때문에 상부 정점으로 하부 정점을 이동시킬 필요가 있다. 단일의 반복 후 위에서 아래로의 길이는 나머지 위치의 길이와 정확히 동일하다는 것을 확신할 수 있다.

```
Vector3 vDelta = constraint.m_vertexTop.m_vCurPos - constraint.m_vertexDown.m_vCurPos;
float fLength = vDelta.length();
vDelta.normalize();
Vector3 vOffset = vDelta * ( fLength - constraint.m_fRestLength );
constraint.m_vertexDown.m_vCurrentPosition += vOffset;
```

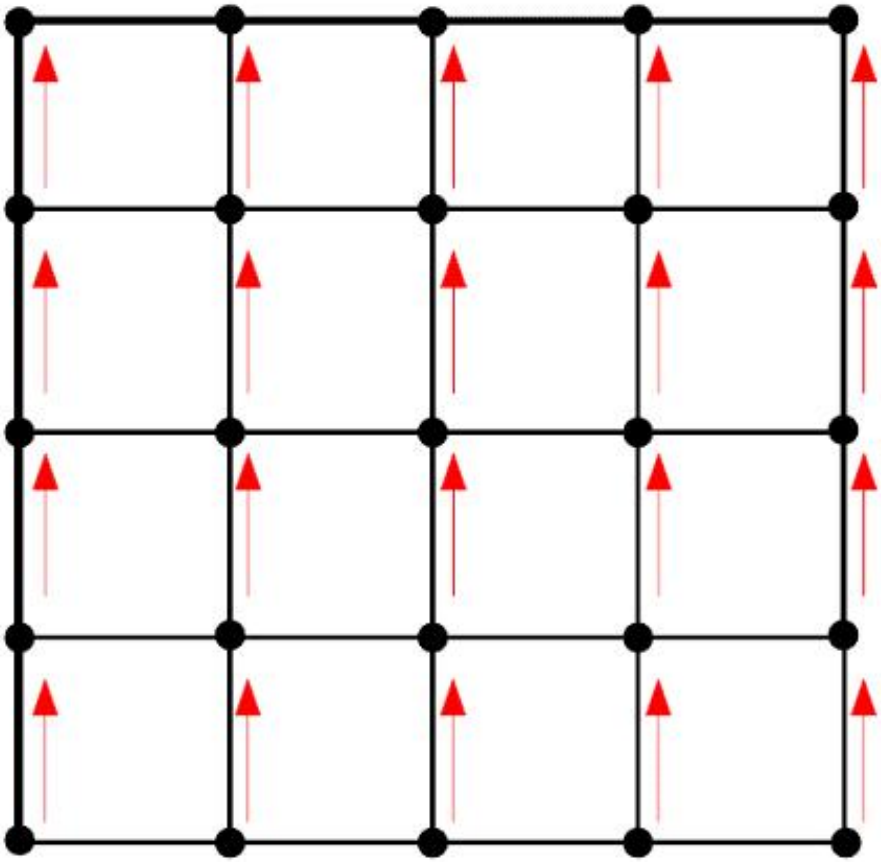

그림 8. 하드 콘센트레이트

보다시피 수평의 재킷 스트레칭을 우리는 고려하지 않고 있다. 하드 콘센트레이트를 수평 방향으로 추가하는 것은 가능하지 않다. 왜냐하면, 이는 두 번 해결될 정점을 야기할 수 있기 때문이다. 즉, 수직 고정 단계의 결과를 잃게 되고 나머지 길이를 더 이상 보존할 수 없다.
아무튼, 재킷에 대한 수평 스트레칭은 실질적으로 육안으로는 파악되지 않지만, 수직 스트레칭은 재킷을 매우 안 좋게 보인다는 것을 알 수 있다. 이 솔루션은 충분했다.

재킷 테두리

우리가 원하는 두 번째 것은 재킷의 테두리를 재킷의 나머지보다 조금 더 움직이도록 하는 것이다. 예를 들어, 공개 재킷으로 구동하는 경우 공기 저항이 재킷의 중심보다 재킷의 테두리에 더 큰 영향을 미친다는 것을 알게 될 것이다. 이는 당신의 몸이 재킷의 나머지 부분에 대해 바람막이 역할을 하게 되는 것이다.
재킷의 테두리는 얼마나 많은 콘센트레이트가 붙었는지 그 수를 세서 쉽게 발견할 수 있고, 4 개 미만의 콘센트레이트가 있는 정점이 테두리이다. 그렇기 때문에, 우리는 이러한 정점을 기준으로 해서 이들을 다음과 같은 상이한 파라미터를 사용해서 시뮬레이션한다.

낮은 댐핑 팩터

글로벌 윈드(global wind)는 큰 영향을 미친다.

World space motion 은 큰 영향을 미친다(아래의 world space motion 과 비교해).

허용된 최대 안전 거리는 더 길다.

테두리의 내부 횟수는 재킷의 나머지 부분과 상이하다. 대신에, 테두리에 충격을 주기 위한 큰 추로서 전체의 재킷이 훌륭한 2 차적인 움직임을 이동으로 추가한다.

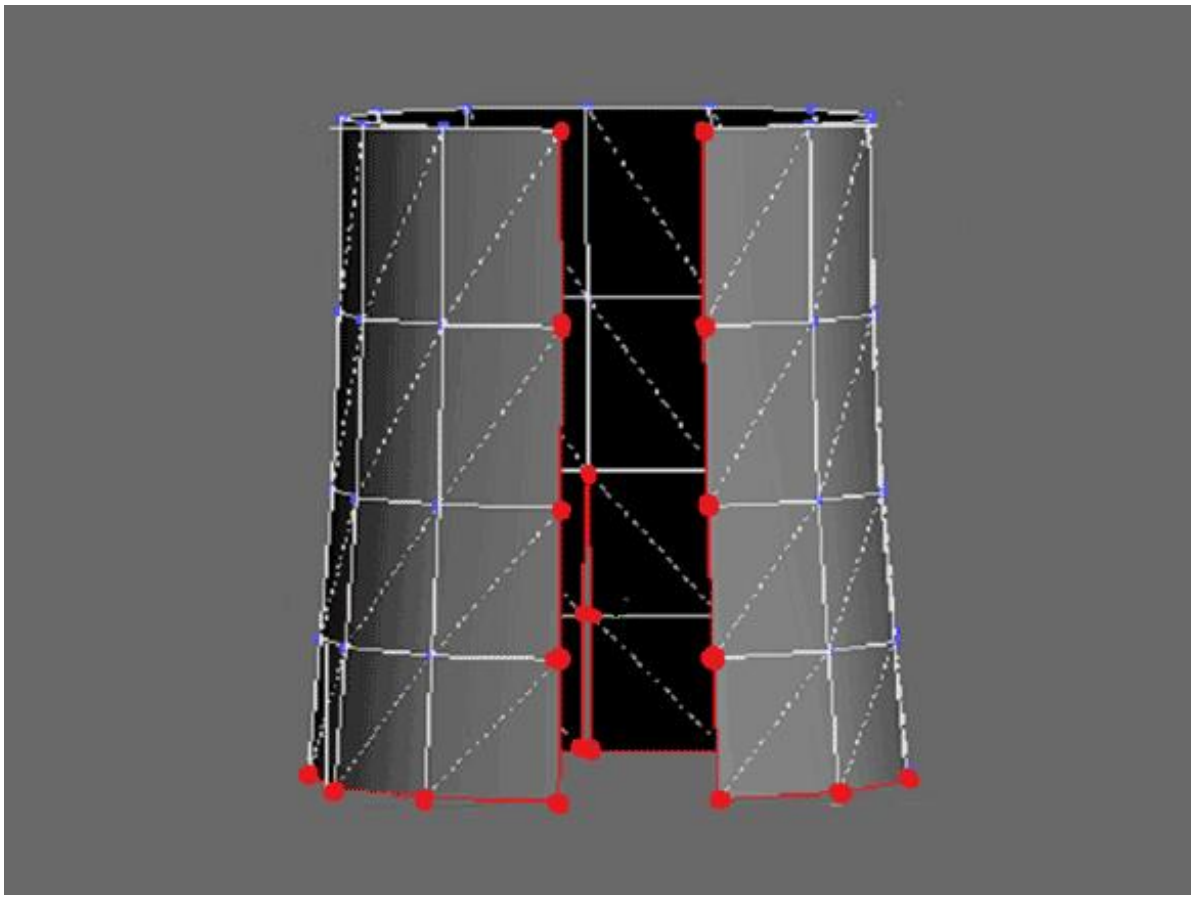

그림 9. 테두리 정점

World Space vs. Local Space 움직임

다음으로 우리가 인지한 바는 캐릭터가 움직일 때 world-space 움직임이 시뮬레이션에 상당히 크게 영향을 미치는 반면에 어깨의 작은 지엽적인 몸의 회전이나 움직임은 알아채지 못한다.

기존의 클로스 시뮬레이션에서 정점의 위치는 world-space 에서 시뮬레이션된다. 누군가는 이는 클로스를 시뮬레이션하는 정확한 방법이라고 주장할 수 있지만 그렇진 않은 것 같다. 그렇기 때문에, 대신에 우리는 캐릭터의 local space 에서 재킷을 시뮬레이션하고 small world motion 을 개별적으로 추가했다. 움직임의 10 ~ 30%와 함께 지엽적인 골격 애니메이션 움직임의 100%는 우리가 찾고 있는 결과를 보여줬다.

마찰

마지막으로 낮은 움직임의 재킷과 높은 움직임의 재킷 간의 대비를 과장한다. 알란이 걸을 때 우리는 재킷이 상당히 정돈되어 있어서 몸을 피하거나 점프하는 경우에 더욱 실감나는 재킷의 움직임을 가질 수 있게 되었다.

재킷이 몸에 붙어있을 때 재킷은 덜 움직여야 한다고 생각했다. 왜냐하면, 재킷과 셔츠의 마찰로 인해 재킷이 올라갈 때 재킷이 자유롭게 움직일 수 있는 만큼 더 큰 움직임이 있어야 하기 때문이다. 우리는 타원체와 닿아 있는 각 정점에 대한 더 높은 댐핑 값을 사용해서 이를 꾸며냈다. 이 방법은 바디와 닿아있는 정점이 약간은 까다로워 보이고 일반적인 상황에서의 재킷과 빠른 동작에서의 재킷 간의 충분한 대비를 제공한다.

결론 및 추가적인 작업

클로스 시뮬레이션의 첫 구체화는 게임 발전 연구에서 “클로스”란 단어에 대해 연구하고 발견되는 알고리즘을 적용함으로 인해 달성하는 것이 매우 단순했다. 품질 바에 충분한 상태로 재킷의 모습과 느낌을 얻는 두 번째 단계는 연구 논문, 수 십 번의 시도와 에러, 그리고 스크랩한 코드의 몇 줄을 통해 브라우징하는 것과 연관되어 있다.

당연히, 여기에는 개선의 여지가 있다. 일례로 고해상도 메쉬로의 매핑에 더해 저해상도 시뮬레이션은 모든 클리핑을 완벽하게 해결하는 것을 어렵게 한다. 제 때 이를 하지 못한 다른 작은 특징은 재킷이 접히는 맵을 주름지게 하거나 토네이도가 와서 재킷이 격렬하게 뒤집힐 수 있다.

결국, 우리의 클로스는 다른 게임의 클로스 시뮬레이션과 명백하게 다르기 때문에 이러한 노력은 대단한 가치가 있다. 이는 실크나 고무라기 보다는 훨씬 트위드같이 보인다. 우리의 설정은 또한 다른 직물(예를 들면, Barry Wheeler 의 다운 재킷)을 시뮬레이션할 때 훨씬 더 유연성이 있고 노파의 베일을 동일한 시스템을 통해 시뮬레이션했다는 것이 증명되었다. 상이한 직물의 외형은 파라미터를 비틀어서 쉽게 이행되었다.

그림 10. 트위드 재킷