
# 수백만을 위한 물리 엔진 확장

(Scaling a Physics Engine to Millions of Players)

작성자 : 케빈 헤(Kevin He)

작성일 : 2012 년 7 월 12 일

[블리자드의 물리 담당이자 지금은 온라인 게임 로블록스의 네트워크 책임자인 케빈이 강체(rigid body) 시뮬레이션의 비밀을 공유한다. 레고 형식의 조립 게임에서 동시에 수백만 명의 플레이어를 수용하면서도 성능과 재미 어느 것도 희생하지 않는 방법을 살펴본다.]

<로블록스(Roblox)>는 종합적인 물리 엔진(physics engine)을 기반으로 하고 있으며, 수백만의 유저들은 그 능력의 한계를 시험하기라도 하듯 계속하여 콘텐츠를 생산해 내고 있다. 우리는 유저들의 무한한 창의력에 부응하기 위해서, 그리고 유저들의 거대하고 복잡한 시뮬레이션에 대한 열망을 충족하기 위해서는 물리 엔진 최적화에 헌신적인 노력을 기울여야 했다.

<로블록스>는 강체에 작용하는 물리 현상을 모두 시뮬레이션한다. 유저들이 벽돌, 블록, 쐐기, 구, 원통 등에 기초하여 스스로 게임을 만들어 내기 때문이다. 이런 요소들은 실제 세계에서와 똑같은 방식으로 행동함으로써 학습곡선을 완만하게 만드는 효과를 가진다.

유저들은 거의 모든 장르의 게임을 만들어 낼 수 있게 되며, 발견 가능성이라는 측면도 더한다. 이런 특징으로 인해 2011 년에만 게임플레이 시간 2 억 5 천만 시간을 달성했다.

복잡한 물리 시뮬레이션이 그 자체로 좋은 게임을 만들지는 못하며, 오히려 자원의 낭비가 될 수도 있다는 점은 짚고 넘어가야 한다. 오히려, 물리 엔진의 복잡성은 게임 플레이의 요구사항에 가깝다고 할 수 있다.

## <로블록스>의 물리학 - 맥락적 접근

<로블록스>는 게임에 있어 물리 복잡성 스펙트럼의 극단적 위치에 자리하고 있다. 다른 쪽 끝에는 <에버퀘스트>나 <월드 오브 워크래프트> 같은 MMORPG 가 있다. 이런 게임들은 기본적인 충돌 검출(collision detection)을 도입해 플레이어들이 유효 영역 밖으로 넘어가지 못하게 한다.

<헤일로(Halo)> 등의 일인칭 슈팅게임, <디아블로 3>와 같은 액션 RPG 는 충돌 검출을 한 단계 더 발전시켜 접촉력과 제한적인 충돌 도형(캡슐과 같은)을 사용해 충돌 반응을 만들어낸다. 이들 게임은 또한 캐릭터에 '수동적' 랙돌(ragdoll) 상태를 도입하여 사망 후의 쓰러짐, 폭발로 인한 반동과 낙하를 랙돌 상태로 처리한다. 그리고 '장식적' 랙돌을 사용하여 - 머리카락의 움직임과 같은 - 동적인 시각효과를 만들어 낸다.

물리 시뮬레이션이 흥미롭고 복잡해지는 시점은 스포츠 게임이다. MMORPG 와 액션게임에서 볼 수 있는 것들에 더하여, <파이트 나이트(Fight Night)>이나 <FIFA 12>같은 스포츠 게임에서는 격투선수와 축구 선수의 관절에 구동부(motor)가 실제 힘을 가하는 '능동적' 랙돌을 도입한다. 이런 방식은 부드럽고 생동감 있는 동작과 애니메이션을 만들 수 있게 한다. 이 방식에서 구동부가 없다면, 격투가와 선수의 사지는 생기없이 매달려 있게 되고, 캐릭터는 지면으로 쓰러져 버릴 것이다.

마지막으로, <로블록스>가 자리한다. 완벽한 물리 시뮬레이션이 기본적으로 활성화되어 있으며, 예외적으로만 비활성화되는 게임이다. 지면을 제외한 모든 물리적 물체가 시뮬레이션된다. 엔진의 기능은 다음과 같다.

- 접촉력이 작용하는 충돌검출, 풍부한 충돌 도형(육면체, 구, 원통, 쐐기, 볼록다면체 외 다수)
- 강체 역학, 단일 파이프라인 힘 적분 (중력, 접촉력, 마찰력, 관절/용수철, 부력, 항력, 운동에너지)
- 물리 기반 캐릭터 애니메이션, "수동" 및 "능동" 랙돌, 균형 제어 (서기, 달리기, 뛰기, 낙하, 수영 지원)
- 기계적 탑승물과 배, 다양한 관절 타입 (운동학, 경첩, 결합, 로프, 다각기둥, 회전, 구동 등)
- 부력과 진보한 액체 역학

이들 기능은 모두 네트워크를 통한 멀티플레이어 환경에서 동작한다.

## 개별에서 총체로

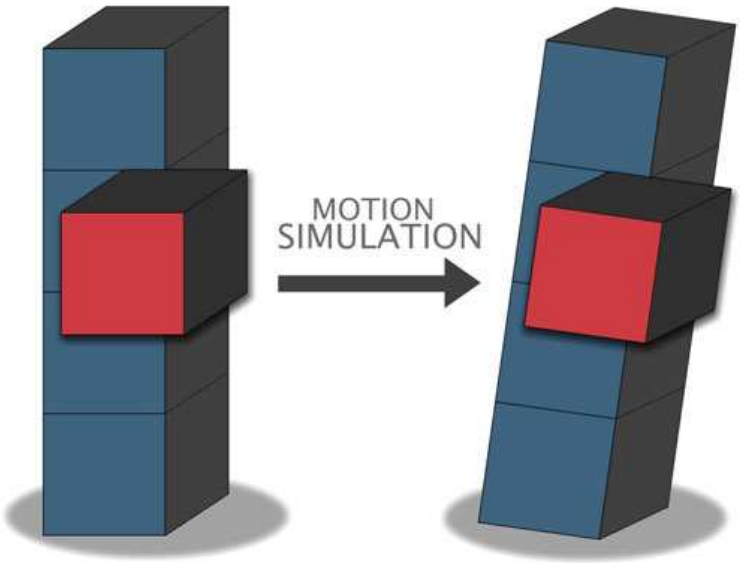
조립 블록을 접한 사람들은 대부분 블록을 최대한 많이 사용해 뭔가 거대한 것을 만들려고 한다. <로블록스>도 예외는 아니어서 고층건물이나 타이타닉 규모의 함선의 구성 부속을 각기 개별적인 강체로 시뮬레이션하려면 네트워크와 CPU/GPU 자원의 소모가 엄청날 것이다.

<로블록스>의 엔진에선 거대한 구조물과 탑승물을 시뮬레이션하는 데 영리한 방법을 사용했다. 고층건물이 지면에 고정되어 움직이지 않는다면, 이 건물은 배경과 결합된 단일 개체로 취급된다. 개별 벽돌에는 동적인 시뮬레이션이 수행되지 않고, 고층건물 전체와 다른 물체와의 충돌 검출만 수행된다. 엔진은 메모리를 적게 소모하도록 전체 고층건물을 정적인 "경량급(featherweight)" 물체로 압축한다. "경량화" 덕분에 최소한의 비용으로 <로블록스> 세계에 수천 개의 복잡한 빌딩을 넣을 수 있었다.

타이타닉 규모의 함선이 항해하는 상황에서는 전체 함선이 동일한 강체 역학 엔진에서 구동되어야 물과의 상호작용 및 다른 부유물과의 충돌을 사실적으로 시뮬레이션할 수 있다. 각 함선의 구성 요소를 개별 강체로 취급하여 시뮬레이션하면 엄청난 비용이

필요하기에, <로블록스> 는 그러한 거대한 결합된 강체는 "어셈블리(Assembly)" [1] 로 시뮬레이션하여 시뮬레이션에 소요되는 자원을 더 효율적으로 관리한다.

<로블록스> 는 한 프레임 동안 서로 상대적으로 움직이지 않는 물체 쌍을 동적으로 식별한다. 그러한 물체는 하나로 그룹 지어 어셈블리를 형성하게 된다. 타이타닉 규모의 함선이 천 여 개 이상의 부속이 결합되어 있을지라도 어셈블리로 구성되어 있다면 단일 강체로 시뮬레이션 한다. 물리 시뮬레이션을 수백만 이상의 요소로 확장하고자 할 때에는 이런 식의 효율성이 필요하다.




**이 경우에 두 요소는 어셈블리를 형성한다. 한 프레임의 동작 동안 그 둘이 서로 상대운동을 하지 않기 때문이다.**

어셈블리가 만능은 아니다. 구현된 어셈블리는 나름의 과제를 안고 있다. 끊임없이 운동, 충돌, 폭발, 인간의 개입이 이루어지는 동적인 환경에서 어셈블리는 언제든지 구성되고, 쪼개지고, 합쳐질 수 있다. 흔히 벌어지는 상황은 다음과 같다.

- 결합된 요소는 충분한 이탈력이 가해지면 분리될 수 있다.

[1] 두 개 이상의 물체가 결합된 구조물을 통칭하는 용어(역주)

- 분리되어 있던 물체는 한쪽 물체의 "입구"와 다른 물체의 "출구"가 접촉하면 런타임에 하나로 결합될 수 있다.
- 인간형 캐릭터가 죽을 때, 검으로 공격당한 경우라 할지라도 관절이 분리되면 신체가 절단될 수 있다.
- 유저가 LUA 스크립트를 사용하여 두 요소 사이에 관절을 추가하거나 제거하여 어셈블리를 동적으로 합치거나 분리할 수 있다.

우리 엔진은 이런 변화가 생길 때 끊임없이 형성되는 어셈블리를 클라이언트와 서버 간에 실시간으로 동기화해야 한다. 이때 클라이언트와 서버가 동일한 물체를 루트(root body)로 선택해[2] 어셈블리를 생성하여 네트워크를 통한 동작을 부드럽게 보이도록 한다.

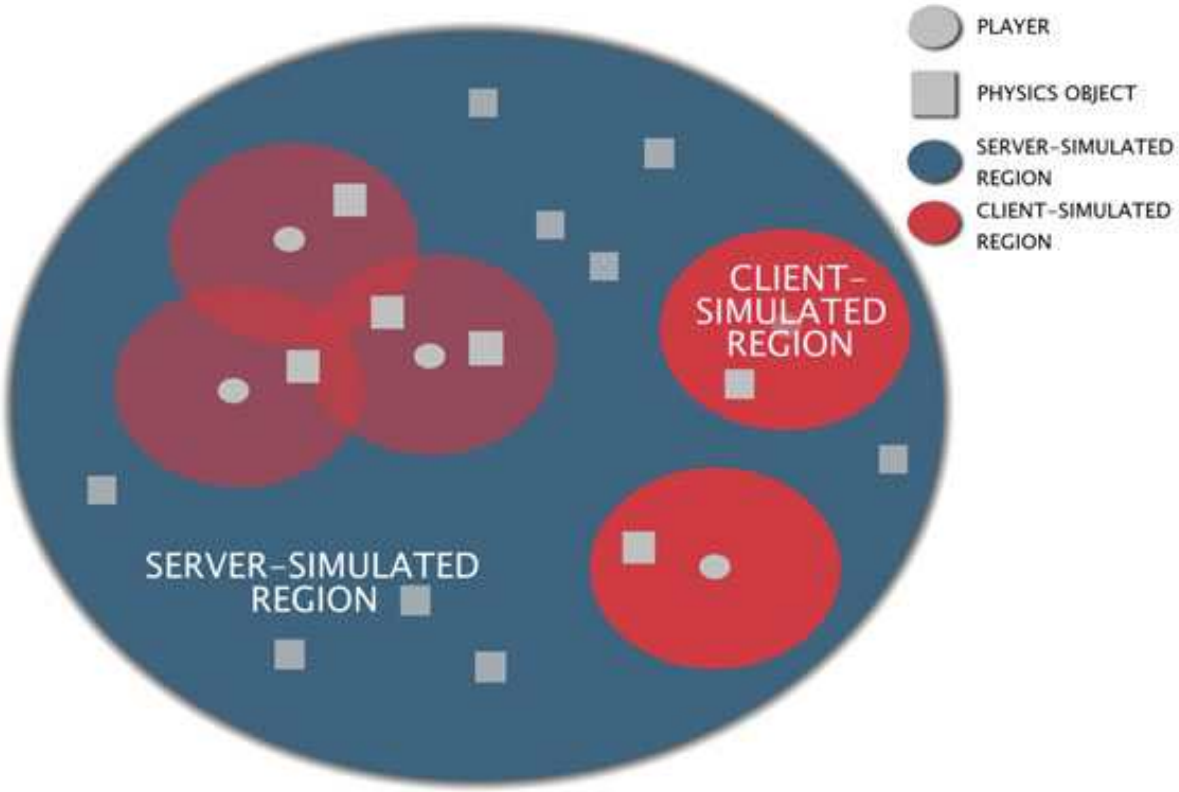
## 중앙 처리에서 분산 처리로

어셈블리와 더불어, 물리 시뮬레이션을 중앙 처리에서 분산 처리로 전환함으로써 더 많은 플레이어와 더 많은 물리 시뮬레이션을 수용할 수 있는 공유 공간을 만들 수 있었다. 초창기에 <로블록스>는 물리 시뮬레이션을 중앙 처리방식으로, 즉 하나의 게임 서버에서 물리 계산이 처리되고 그 결과가 플레이어에게 전달되어 화면에 표시되는 모델을 사용하였다.

작은 규모, 즉 적은 수의 플레이어, 물리 연산, 시뮬레이션 요소들만 있는 상황에서는 중앙 처리 방식이 잘 작동한다. 하지만 <로블록스> 에서 구현된 중앙 처리 모델로는 게임의 성장과 함께 놀랍도록 복잡해진 물리 연산을 감당할 수 없었다. 서버가 너무 많은 요소와 플레이어의 조작을 시뮬레이션해야 하는 상황이 오자 움직임이 실시간으로 돌아가지 않게 되었다.

이 문제는 2010 년에 중앙 처리 엔진을 분산 물리 엔진으로 발전시킴으로써 해결되었다.

[2] 어셈블리를 구성할 때 트리 형식으로 표현하는데, 그때 트리의 루트를 동일한 물체로 선택한다는 뜻(역주)




**이 그림은 분산 물리 엔진 모델을 간략히 보여준다. 실제로는 클라이언트에서 시뮬레이션되는 영역의 모양은 항상 동일하지 않다.**

분산 물리 연산은 물리 시뮬레이션을 서버와 플레이어 모두에게 분배한다. 위 그림에서 파란 영역은 서버(서버 시뮬레이션 영역)와 물리적 물체 및 플레이어 전부를 포함한다. 어떤 영역 안에 플레이어가 전혀 없을 때는 서버에서 물리 시뮬레이션을 처리하여 플레이어게 전달하는데, 여기까지는 중앙 처리 모델과 같은 방식이다.

차이점은 각 플레이어가 물리적 대상의 동적 영역을 가지고 있다는 점이다. 각 플레이어가 근접한 영역(클라이언트 시뮬레이션 영역)의 시뮬레이션을 로컬에서 수행하여 서버의 부하를 줄여준다. 이러한 분산 아키텍쳐로 거의 무한정에 가까운 수의 물체에 작용하는 물리 현상을 성능 저하 없이 시뮬레이션 할 수 있게 되었다.

여기에 로드 밸런싱 알고리즘을 구현하여 각 플레이어의 연산 능력과 대역폭에 따라 영역 크기를 늘리거나 줄일 수 있도록 하였다. 하드웨어나 인터넷 회선이 느린 플레이어는

일반적으로 더 적은 수의 물체를 시뮬레이션 하게 되며, 빠른 회선과 최신 장비를 갖춘 경우는 그 반대이다. 분산 물리 연산의 핵심 과제는 두 플레이어 영역의 경계에 있는 물체를 시뮬레이션하는 방법이다.

매우 단순한 상황에서는 명료한 분할이 잘 작동한다. 예를 들어, 한 물체가 한 플레이어의 시뮬레이션 영역에만 속해 있고, 다른 물체와는 상호작용하지 않는 경우가 그렇다. 이 때 플레이어는 강체 시뮬레이션을 수행하고 다른 플레이어에게 그 결과를 전송한다.

더 어렵고, 더 자주 발생하는 시나리오는, 물체 A 와 물체 B 가 서로 충돌하는데 각 물체가 각기 다른 플레이어의 영역에 속하는 경우이다. 이론적으로는, 한 플레이어가 물체 A 를 시뮬레이션하고, 다른 플레이어가 물체 B 를 시뮬레이션한 후에, 그 결과를 서로 교환하면 된다. 네트워크를 통한 갱신이 즉시 이루어지는 상황에서는 이런 방식이 가능하기는 하다. 하지만 렌더링 사이클(이를테면 30hz)에 비해 네트워크 갱신 주기는 대역폭을 확보하기 위해 대체적으로 긴 편(이를테면 15hz)이다. 물리 엔진의 시뮬레이션이 네트워크 갱신 주기의 두 배인 30hz 로 동작한다는 가정을 하면, 물리 시뮬레이션을 한번씩 건너 뛰고 네트워크 갱신이 이루어진다. 고속 충돌 상황에서는 이 부분이 문제가 될 수 있다.

예를 들어 보면,

A: 플레이어 A 에 의해 시뮬레이션 되는 차량

B: 플레이어 B 에 의해 시뮬레이션 되는 벽돌

시나리오: 차량 A 가 벽돌 B 를 향해 달려간다. 차량 A 는 탄성 벽에 부딪힌 후 튕겨져 나온다. (더 자세한 것은 이 현상에 대해 쓴 이 글을 참고할 것) 차량 A 의 속도는 늦춰지고 벽돌 B 는 떨어져 나온다. 이 경우에 오류가 생기는 과정은 다음과 같다.

**시뮬레이션 A (차량 A/플레이어 A)**

A 가 B 와 빠른 속도로 부딪히며 접촉력을 만들어내고, B 에서 A 가 떨어진다.

A 가 튕겨져 나오고 B 로부터 멀어지며, A 의 정보를 B 로 전송한다.

A 가 B 의 정보를 전송 받은 결과, B 는 움직이지 않는다.

**시뮬레이션 B (물체 B/플레이어 B)**

B 가 A 의 정보를 전송 받을 때, A 는 B 보다 두 단계 앞서있다.

A 는 이제 한 단계 앞서 있게 되며, B 의 정보를 A 에게 전송한다.

B 가 A 의 정보를 전송 받는다. A 는 B 로부터 멀어지고 있고, B 는 움직이지 않는다.

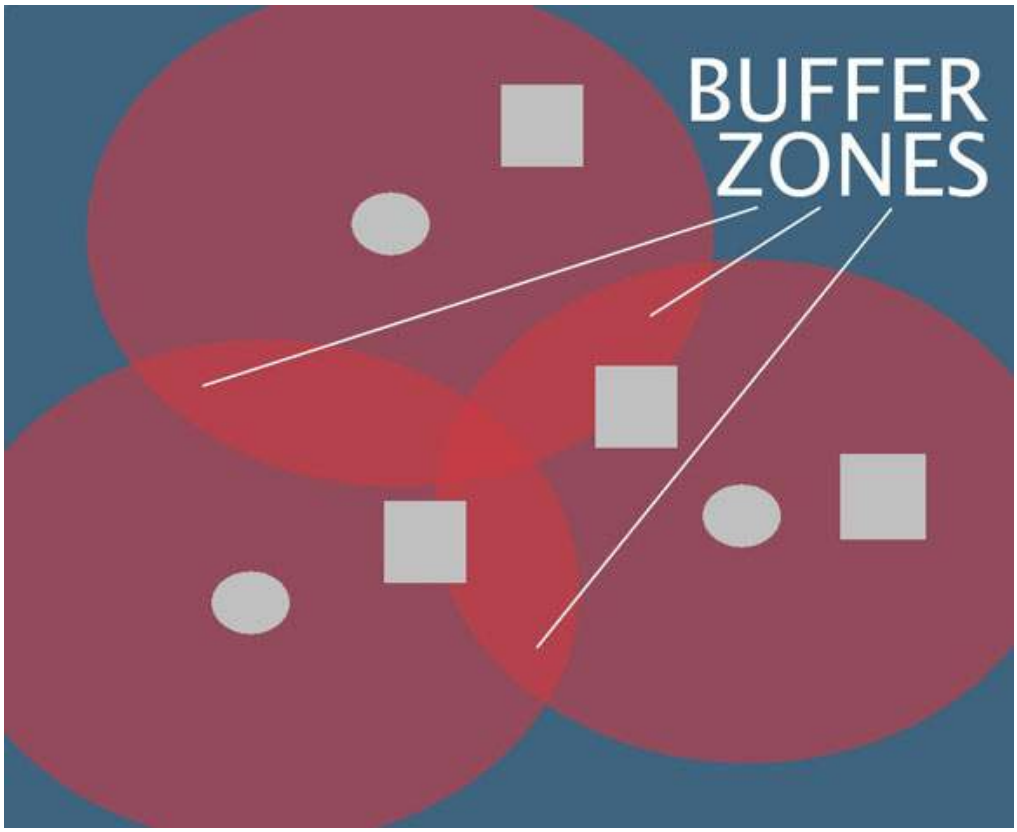
시뮬레이션 B 에서는 시뮬레이션 A 가 충돌을 처리하고 차량과 벽돌을 떨어뜨리게 하기 전에 충돌을 감지할 기회가 없다. 결과적으로, 벽돌 B 는 고정되어 있는 것처럼 행동하게 되고, 차량 A 만이 튕겨져 나온다.

저속 충돌이나 경계간 조작에서는 이런 상황은 발생하지 않는다. 양 쪽 모두 상호작용을 감지할 기회가 있으므로 신뢰할 만한 반응을 만들어 내기 때문이다. 네트워크 갱신 주기를 짧게 하거나 네트워크 지연을 줄이면 고속 충돌의 결과를 향상시킬 수는 있으나 문제가 완전히 해결되진 않는다. 충돌에 관여하는 두 물체가 같은 시뮬레이터에서 시뮬레이션되고 있지 않기 때문이다.

## 분산 물리 연산과 버퍼 구역

<로블록스>는 경계간 시뮬레이션 문제를 두 플레이어의 시뮬레이션 영역 사이에 "버퍼 구역(buffer zone)"을 만들어서 해결한다. 버퍼 구역 안에 있는 물체는 양쪽 플레이어 모두에 의해 시뮬레이션 되는 것이다. 위의 차량 충돌 문제에서, 차량과 벽돌은 플레이어 A 와 플레이어 B 양쪽 모두에게서 시뮬레이션 된다. 그 결과 이 데모 영상과 같이, 차량 A 와 벽돌 B 는 충돌 상황에서 올바른 행동을 보이게 된다.

버퍼 구역이 활성화되어 있으면 두 플레이어는 동일한 물리적 물체(들)을 시뮬레이션한다. 이때 한 플레이어가 "권한(authority)"을 가지며, 물리 시뮬레이션을 원격으로 수행하여 그 결과를 서버를 통해 다른 플레이어에게 전송할 책임을 갖는다. 다른 플레이어는 "종속성(secondary)"을 갖게 되고, "권한"을 가진 플레이어로부터 정보가 오기 전까지의 자료 공백을 메우는 용도로만 로컬 시뮬레이션을 수행한다. 이때 <로블록스>는 원격 시뮬레이션(실제)과 로컬 시뮬레이션(예상)을 블렌딩해 준다.




**붉은 영역이 겹쳐진 부분이 버퍼 구역을 의미한다. 버퍼 구역의 물체는 양쪽 플레이어 모두가 시뮬레이션을 수행한다.**

가장 단순한 블렌딩 알고리즘은 원격과 로컬 시뮬레이션의 결과를 섞는 것이다. 원격 정보가 도착하면 물체의 위치를 변경하는 식이다. 이런 방식은 작동하긴 하지만 원격 시뮬레이션과 로컬 시뮬레이션 간의 불일치 때문에 버퍼 구역에서 고속으로 이동하는 물체는 끊겨 보일 수 있다.

이런 불일치는 존재할 수 밖에 없으며, 그 양은 네트워크 지연과 비례한다. 로컬/원격 물리 시뮬레이션 간의 네트워크 지연은 각 네트워크 패킷에 시뮬레이션 되는 물체의 속도/위치 정보와 타임스탬프(발신/수신 시각)를 넣어 해결할 수 있다. 원격 시뮬레이터에서 보낸 ( t2 시점에서의) 정보는 다음과 같은 식이다.

- 발신 시각 : t1
- 위치 : x
- 속도 : v

x(원격 시뮬레이션 결과 상의 위치)를 로컬 시뮬레이션의 결과와 직접 블렌딩하지 않고, 네트워크 지연 문제 해결을 위해 x 를 다음과 같이 계산한다.

$X_{원격}' = X_{원격} + (t2 - t1) * v$

그 후에, $X_{원격}'$을 로컬 시뮬레이션의 결과인 $X_{로컬}$과 블렌딩한다. 재계산한 원격 시뮬레이션 결과와 로컬 시뮬레이션 결과 간의 불일치는 등속 운동하는 물체에 대해서는 거의 0 이 된다. $X_{원격}'$과 $X_{로컬}$의 차이가 매우 작으면, 움직이는 물체의 궤적을 렌더링할 때에 $X_{로컬}$을 직접 사용해도 끊김 현상이 나타나지 않는다.

물체가 충돌하거나, 현저한 속도변화가 있을 경우, 속도 값은 매우 짧은 시간 동안만 유효하기 때문에 재계산한 원격 시뮬레이션 결과가 소용 없는 경우가 생긴다. 이런 상황에서는 로컬 시뮬레이션에 더 많은 가중치를 주고, 충돌이 안정화 되거나 가속도가 0 이 되는 시간 동안 점진적으로 원격 시뮬레이션 결과의 가중치를 늘려 간다. 관련 충돌로부터 경과한 시간이 e 초라고 할 때, 원격 시뮬레이션과 로컬 시뮬레이션 사이에서 위치 x 는 다음과 같이 블렌딩할 수 있다.

e < 1 일 때:

$X = (1 - e) * X_{로컬} + e * X_{원격}'$

e > 1 일 때:

$X = X_{원격}'$

위 수식을 사용함으로써 두 시뮬레이션을 보충할 때 물체가 끊겨 보이는 현상이나 부자연스러운 움직임이 일어나지 않도록 물체의 운동을 충분히 정확하게 예측할 수 있다.

그러나 이 방식도 완벽하지는 않다. 향후 목표는 별개의 물리 시뮬레이션 사이에서 일어나는 운동을 블렌딩하는 보다 지능적인 알고리즘을 만드는 것이다. 궁극적으로는 수백 명의 플레이어에게서 시뮬레이션된 결과를 매끄럽게 연결하여 거대하고, 일관성 있고, 물리적으로 정확한 세계를 렌더링할 수 있는 분산 물리 엔진을 만들고자 한다.

## <로블록스> 물리학의 진보

<로블록스> 의 물리학 엔진이 다룰 수 있는 규모를 확대하고 새 역학 모델을 도입하는 작업은 끊이지 않고 있다.

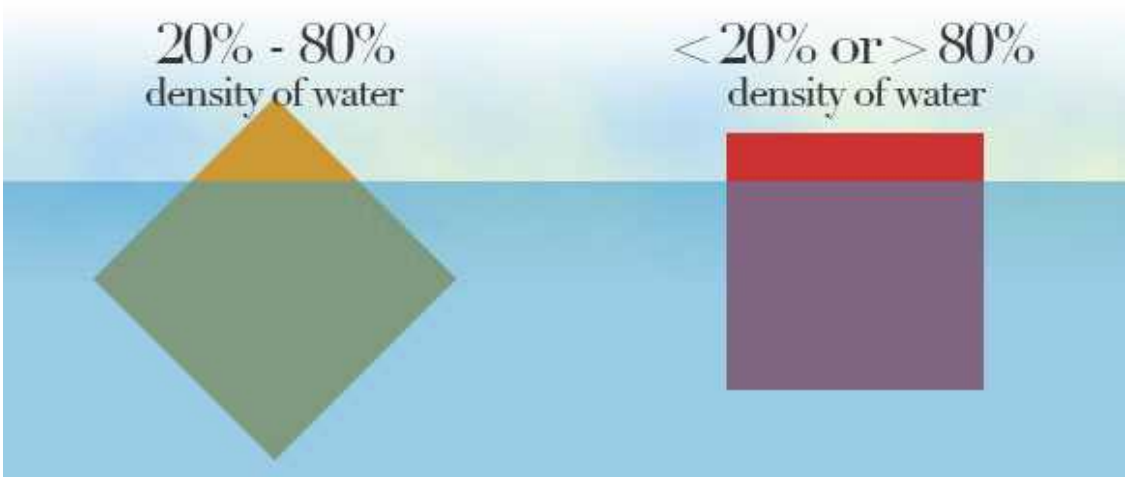
예를 들어, 최근에는 물과 부력 시뮬레이션을 <로블록스>에 추가하였다. 구현의 주안점은 간결한 구조를 통해 물 속 물체와 캐릭터의 행동이 <로블록스> 물리학을 자연스럽게 확장한 것처럼 보이고 느껴지도록 만드는 것이었다. 이는 표준 물리 파이프라인에 부력이 포함되어야 함을 의미한다. <로블록스>에서는 물이란 대상을 물리적 물체로 간주하고, 물과 다른 물체 사이의 접촉력(또는 충돌)을 통해 부력이 작용한다. 물과 부유물, 부유물과 부유물, 부유물과 침강물 사이의 충돌은 모두 통합 엔진으로 시뮬레이션 된다.

<로블록스>의 엔진은 모든 힘이 - 부력, 항력, 유수력, 중력, 접촉력, 그 밖의 모든 외력 - 통합되어 있다. 부유물의 움직임은 뉴튼의 법칙을 따르는 강체에 작용하는 알짜 힘의 결과이다.

또한 물과 부력 시뮬레이션은 높은 수준의 디테일을 갖도록 구현되었다. 예를 들어, 정육면체를 물에 떨어뜨린다면, 중력 때문에 물 속에 잠기고 나면, 부력이 수면으로 되돌려 보내는 힘을 가함에 따라 이 정육면체는 흔들리고 방향이 바뀐다. <로블록스>에서는 정육면체를 따로 8 개의 복셀(voxel)로 분할하고, 각각이 계산된 부력을 보간하도록 하여 이러한 방향 전환을 시뮬레이션한다.

부유물(육면체, 쐐기꼴, 직각 쐐기꼴)의 부력을 복셀화함으로써 안정화 된 후에 부유물은 올바른 방향을 향하게 되며, 구석에 서있는 플레이어가 가하는 압력 같은 비대칭 외력에 정확하게 반응한다. 또한 물의 항력을 직선운동과 회전운동 모두에 적용하여 운동이 감쇄하도록 하여 현실성을 강화하였다.

또 다른 예를 들면, 이 엔진의 진보한 액체 물리학은 단면적과 겉면적을 정확히 구분하여 부유물에 작용하는 항력을 계산한다. 이 데모 영상에서는, 보트에 힘이나 속도를 지정해주는 속임수를 쓰지 않았다. 하지만 엔진이 노의 구동과 수면의 실제 상호작용을 시뮬레이션하여 보트가 추진되고 있다. 또한 두 배의 배수량이 동일함에도 불구하고 얇은 보트가 두꺼운 보트보다 더 빠르게 나아가는 것을 볼 수 있다. 또한 보트는 물 속의 다른 물체와 상호작용하며, 이는 진보한 액체 물리학과 강체 역학이 통합된 시뮬레이션임을 보여주는 예라고 할 수 있다.




**실제 세계에서와 같이, <로블록스>에서 복셀 육면체의 최종 방향은 밀도의 영향을 받는다.**

이 엔진의 물과 부력 시뮬레이션은 보트 모델링에서 실물 수중 프로펠러에 이르기까지, 수많은 실질적 응용 사례를 찾을 수 있다. 일단 이걸 만들어낸 건 개발자들이지만, 유저들이 이 원시적인 설계를 향상할 방법을 찾아내 주리라 믿는다.

## 이후의 발전

<로블록스>는 사실적인 종합 물리 엔진을 기반으로 만들어졌다. 사실성과 종합성이 플랫폼의 가장 기본적인 요소이며, 이를 통해 게임플레이와 게임 창작 체험에 융통성과 직관성을 부여할 수 있다. <로블록스>는 어셈블리와 분산 물리 연산을 기반으로 액체와 부력의 물리학을 도입하여 현저한 진보를 보였다. 하지만 궁극적 목표는 <로블록스>의 물리 엔진을 수백 명의 플레이어와 수백만 이상의 요소 사이의 상호작용을 다룰 수 있도록 최적화하는 것이다.

물리학 시뮬레이션이 그 정도까지 발전하면 <로블록스> 유저는 그들이 원하는 무엇이든 MMO 세계에서 만들어 낼 수 있게 되고, 꿈과 현실이 만나는 경험을 할 수 있을 것이다.

그때까지는, 진보는 계속된다.