ru※ 본 아티클은 CMP MEDIA LLC와의 라이선스 계약에 의해 국문으로 제공됩니다

Gamasutra.com

대역폭 제한 하에서의 비주얼 엔티티를 전달하기

Olivier Cado
2007년 5월 24일

http://www.gamasutra.com/view/feature/1421/propagation_of_visual_entity_.php

1. 들어가며

*사가 오브 라이즘(Saga of Ryzom)*은 지속성을 지닌 대규모 다중접속 온라인 게임(MMORPG)으로 2004 년 11 월 유럽과 북미에 출시되어 현재까지 3 개 국어로 로컬라이즈 되었다. 2000 년부터 Nevrax에서 개발을 시작하였으며, 2006 년 말 Gameforge에 인수되었다.

Nevrax 팀은 Nevrax Library (NeL)를 이용해 *라이즘*의 개발을 시작해나갔으며 *라이즘*은 GPL하의 무료 소프트웨어로 만들어졌다. NeL에는 몰입도 높은 가상 세계를 다루기 위한 서버 기술이 제작되어 얹혀졌다. 이 글은 동일한 가상 세계 내에 접속된 최종 유저 클라이언트의 아바타를 표현하는 동적 물체인 움직이는 엔티티의 부드러운 동작을 보여주기 위해 개발된 네트워크 기술에 중점을 맞추고 있다.

필자는 이 글을 검토해보고 훌륭한 지식을 전달해주었고, Nevrax 와 Gameforge France 의 CTO 인 Daniel Miller 에게 감사의 인사를 전하고 싶다.

작업의 목적

우리의 클라이언트 소프트웨어는 애니메이션 된 3D 화면이 움직이는 것과 애니메이션 된 엔티티를 표시해야 했다. 플레이어는 자신들의 아바타에 대해 직접적인 조작을 할 수 있어야 하는데, 포인트 & 클릭 방식(주로 2D 나 ¾ 시점의 게임이 채택하고 있지만 3D 게임에서도 간간히 쓰인다.)은 몰입도가 너무 낮은 것으로 판단되었다. 비록 롤플레잉 게임이 FPS(먼저 쏘는 사람이 살아남는다.)나 스포츠 게임(타이밍을 맞추는 것이 전체적인 게임에 매우 중요하다.)처럼 플레이어의 빠른 입력을 필요로 하진 않으나 이동이 조작과 일치하고 부드러울 필요는 있다.

우리는 에닡티의 순간이동이나 위치의 갑작스러운 이동(jolt; 졸트)이 다른 MMOG 보다 적기를 바랬지만 *라이즘* 프로젝트의 시작 당시에는 엔티티가 되돌아가거나 앞으로 뛰는 것으로 보이는 경우가 자주 있었다. 또 다른 중요한 요소는 바로 캐릭터가 심리스(seamless) 세계에서 미리 분할된 세계를 이동하며 로딩 없이 마음껏 움직일 수 있게 하는 것이었다.

이 글에서는 우리가 시도해보고 최종적으로 선택한 기술들을 설명한다. *비쥬얼 속성(visual property)*라는 단어는 역동적인 상태 하에 있는 플레이어나 컴퓨터 조작 하의 엔티티를 뜻한다. 예를 들자면 현 위치와 이동 상황이라던가, 3D 모델 아이덴티파이어(identifier), 옷 모음, 현재 애니메이션 등을 뜻한다. 우리는 클라이언트 소프트웨어가 3D 모델이나 텍스쳐, 애니메이션 등의 데이터에 접근할 권한이 있다는 것을 가정하고 있다. 이 글의 대부분에서 우리는 위치의 이동에 중점을 맞출 것인데, 왜냐하면 이동 경로의 연속적 특징은 부드러운 이동이 이루어지지 않을 때 이것이 눈에 잘 띄도록 만들어주기 때문이다.

도전

온라인 게임에서는, 특히 대규모 다중접속 온라인 게임에서는 대역폭 제한이 모든 플레이어의 위치 변화를 전부 전송하는 것을 불가능하게 만든다. *라이즘*의 출시 당시에는 56k 모뎀이 플레이어들 사이에 흔히 퍼져있었던 때였고, ADSL 은 여전히 걸음마 단계에 지나지 않았다. 우리의 MMORPG 는 56k 모뎀에서 잘 작동해야만 했으며, 심지어 14k 모뎀에서도 작동하게 만들 필요가 있었다.

온레인 게임에서 치팅은 피해자(불이익을 받는 플레이어)의 게임 경험을 완전히 망가뜨리므로 반드시 방지해야 할 필요가 있다. “클라이언트를 믿지 말라”는 유명한 격언이 있는데, Nevrax 의 창립자들은 *라이즘*의 클라이언트를 무료 소프트웨어 라이선스로 공개하길 원하였기에 *라이즘*의 클라이언트는 오로지 디스플레이와 입력 기능만을 포함시키고 그 외의 것은 전부 서버에 올리는 형태를 취해야만 한다는 사실이 금방 명백해졌다.

게임 정보를 서버 측에서 다루는 것은 플레이어가 제작한 레이더 같은 것을 방지할 수 있다. 그리고 P2P(peer-to-peer) 방식의 네트워크 솔루션을 배제함으로써 대역폭 제한 문제는 낮은 대역폭을 지닌 사용자의 집뿐만 아니라 우리가 빌릴 예정이었던 서버 팜에서도 발생하게 되었다.

MMORPG를 플레이하면서 따라오는 문제는 바로 렉(lag)이다. 렉은 행동과 화면 사이에 딜레이가 생기는 불쾌한 현상으로 애니메이션이 멈춰버리거나 마구 끊기는 등의 형태로 나타난다. 이러한 렉 현상은 디자인을 통해 최소화시킬 필요가 있다. 렉은 대개의 경우 빡빡한 대역폭 제한 아래의 부적절한 정보 시스템과 인터넷 같은 레이턴시에 취약한 네트워크로 인해 발생하며, 서버의 CPU 처리의 지연으로도 발생한다.

그러므로 우리가 해야 할 작업은 바로 비쥬얼 속성의 전송과 관련된 알려진 지식들을 습득하고 더 나은 것을 개발하는 것이 되었다. 많은 시간을 요구하는 피크를 피하기 위해 CPU 의 퍼포먼스 역시 매우 중요하게 작용하였다.

2. 데드 레커닝(Dead Reckoning)
- 보외법(Extrapolation)

Distributed Interactive Simulation (DIS)와 같은 초창기에 배포된 시뮬레이션 프로젝트는 높은 관성력을 지니는 움직이는 물체를 다루는데 적합하다. 이 비행선 시뮬레이션 환경에서는 일정한 직선적 움직임이 정상적인 것이며, 여기에 점진적인 변화가 일어나게 된다. 그러므로 움직이는 엔티티를 복사해 시뮬레이션해 행동을 복제하는 것이 가능하며, 여기에 행동 변화 이벤트를 적용할 수 있다.

예를 들어 만약 움직이는 엔티티가 s 만큼 속도를 줄인다면 똑같이 s 만큼 속도를 줄이라는 갱신사항이 복제품에게 전달되게 된다. 물론 이러한 갱신은 움직이는 사람(actuator)으로부터 관측자(observer)에게 전달되며, 이 과정에서 상당한 시간을 필요로 할 수도 있다. 복제품의 위치는 갱신이 이루어지기 전까지는

추정(extrapolated)되어 있으며 갱신이 이루어진 후에는 위치가 수정되게 된다. 그러면 이로 인해 복제품에서 보여지는 경로는 실제 경로와는 약간의 차이가 난다.

이러한 위치 오류는 갱신 사항에 추가 정보를 포함시켜서 문제의 정도를 낮출 수 있다. 예를 들어 어느 지점에서 속도 변화가 이루어졌는지를 알리는 타임스탬프가 있다면 갱신 메시지가 전달되는 사이에는 서로 다른 경로가 표시되는 순간이 존재하더라도 최종적으로는 복제된 경로가 원래의 경로와 합쳐지게 된다.

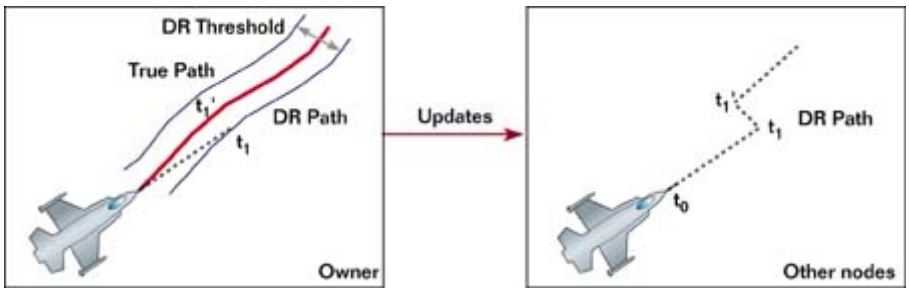

그림 1: 데드 레커닝(Dead Reckoning): 복사된 경로가 원본 경로와 너무 크게 차이 날 경우에 갱신을 실시한다. [Aronson]

그러므로 우리는 부정확한 임시 경로를 정확한 경로와 혼합시킬 필요가 있다. 이것의 결과는 혼합 정도에 따라 달라지게 된다.(앞서 받은 위치들의 기록을 사용한다. [Bernier])

잦은 업데이트를 방지하기 위해서 실제 경로와 복제된 경로가 정해진 수치를 넘어갈 정도로 어긋날 경우에만 갱신을 이루어내도록 하는 것이 가능하다.(그림 1) (이것은 마스터 엔티티와 마스터 프로세스로 복사된 엔티티를 모두 실행시켜 비교해볼 필요가 있을 수도 있다.) 이것이 바로 데드 레커닝(dead reckoning) 시스템이다.(그림 2 의 프로토타입 스크린샷 참고) 데드 레커닝이라는 이름은 고대의 항해사들이 별이 보이지 않을 때 사용한 기술을 따라 붙여진 이름이다.

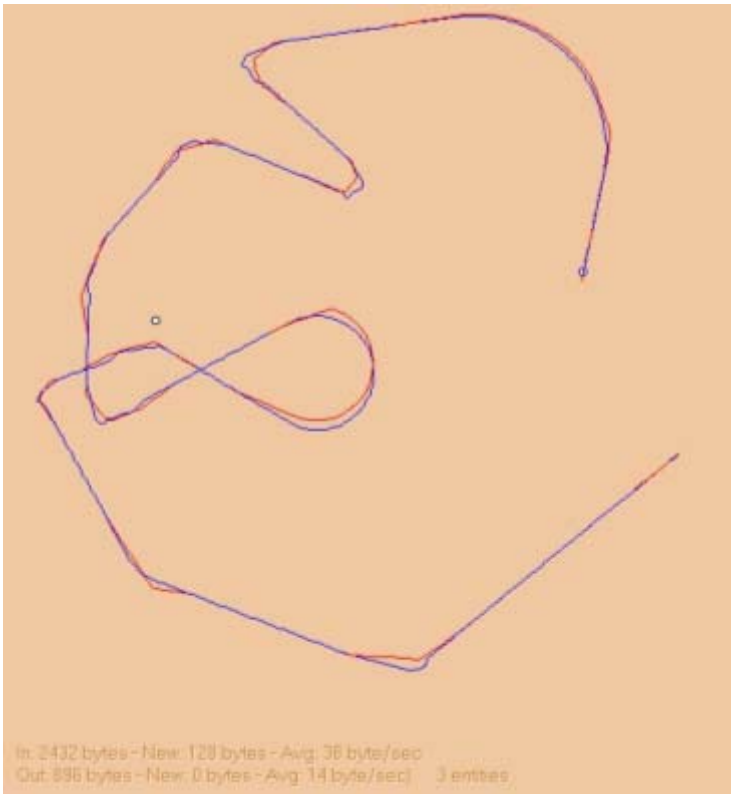

그림 2: Nevrax 에서 2000 년 초 제작한
2D 데드 레커닝의 프로토타입.
원본과 복제 경로가 잘 맞는 것을 볼 수 있다.

이 시스템은 UDP 와 같이 신뢰도가 떨어지지만 경량(lightweight) 네트워크 프로토콜에도 대응할 수 있도록 만들어진다. 이러한 것은 변화가 없다고 하더라도 지속적으로 '연결 유지 (keep-alive)' 갱신을 하기 때문에 메시지가 누락된 경우에도 최종적으로는 모든 옵저버가 최신 정보를 획득할 수 있게 해준다. 이러한 연결 유지 방식은 새로운 접속 클라이언트에 알려진 엔티티를 부여하는 것에도 유용하며, 엔티티의 접속 종료를 알리기도 한다.(연결 유지가 일정 시간 동안 이루어지지 않으면 접속 종료로 간주한다.)

하지만 대부분의 MMORPG 에서는 주로 조작하는 아바타가 큰 관성을 지니는 움직임이 아니라 무작위적인 움직임을 하는 인간형이다. *데드 레커닝*을 사용하는 것은 곧 지속적인 수정 사항의 갱신으로 이어지며 네트워크 트래픽과 눈에 띄는 졸트가 발생한다.(혼합된 복제 경로가 워본 경로와 너무 많이 다를 경우에는 빠른 위치 이동이 필요하기 때문이다.)

그렇다고 하더라도 *데드 레커닝*은 일정한 움직임을 보이는 탈 것이나 AI 엔티티에는 여전히 유용하다.(그렇지만 AI 엔티티가 너무 일정한 움직임을 보이는 것은 결코 바람직하지 않다. 너무 부자연스러운 느낌이 들기 때문이다.) 다수의 플레이어 조작 엔티티를 도입하기 위해서는 다른 전략을 사용할 필요가 있다. "좋은 오래된(good-old)" 위치를 갱신하되 가상 시공간(virtual time space)에서 자연스러운 움직임을 시뮬레이션하고 업데이트 주기의 전체적인 통제를 유지하는 것이 그것이다.

3. 가상 시공간 - 보간법(Interpolation)

2000 년 게임 개발자 컨퍼런스에서 Valve 의 Yahn W. Bernier 는 *하프라이프(Half-Life)*와 *팀 포트리스(Team Fortress)*의 네트워킹에 대한 프레젠테이션을 진행하였다. 그가 내놓은 구상은 바로 미래를 예측(*데드 레커닝*의 원리)하는 대신 과거를 현재처럼 보이게 하는 것이 어떠냐는 것이었다. 클라이언트가 해당 장면 내의 모든 엔티티의 갱신사항을 받은 후라면 정확하고 부드러운 화면을 출력해낼 수 있다는 것이다.

이것이 바로 *렉 보상 시간(Lag Compensation Time)*(LCT)이라 불리는 지연(delay)이 실제 행동과 출력 화면 상에서 발생하는 이유이다. LCT가 높으면 높을수록 갱신되는 정보의 수가 증가하며 이동은 더욱 부드러워진다. 만약 다음 위치를 서버로부터 받기 전에 아바타가 한 위치에 도착하게 된다면 그곳에 멈춰서 위치를 받기까지 기다려야 할 것이다.

이것은 어색하게 보이는 움직였다가 멈추는(start-stop) 모습의 이동을 유발하며, 특히 갱신 주기가 변하는 위치에서는 이러한 현상이 더욱 심하다. 갱신 주기의 변화는 인터넷을 통하는 경우 어쩔 수 없는 현상이다. LCT의 목표는 이것이 일어나지 않게 하는 것이지만, LCT가 렉이나 일시적 불일치를 나타내는 것이니만큼 LCT는 최대한 작아야만 한다.

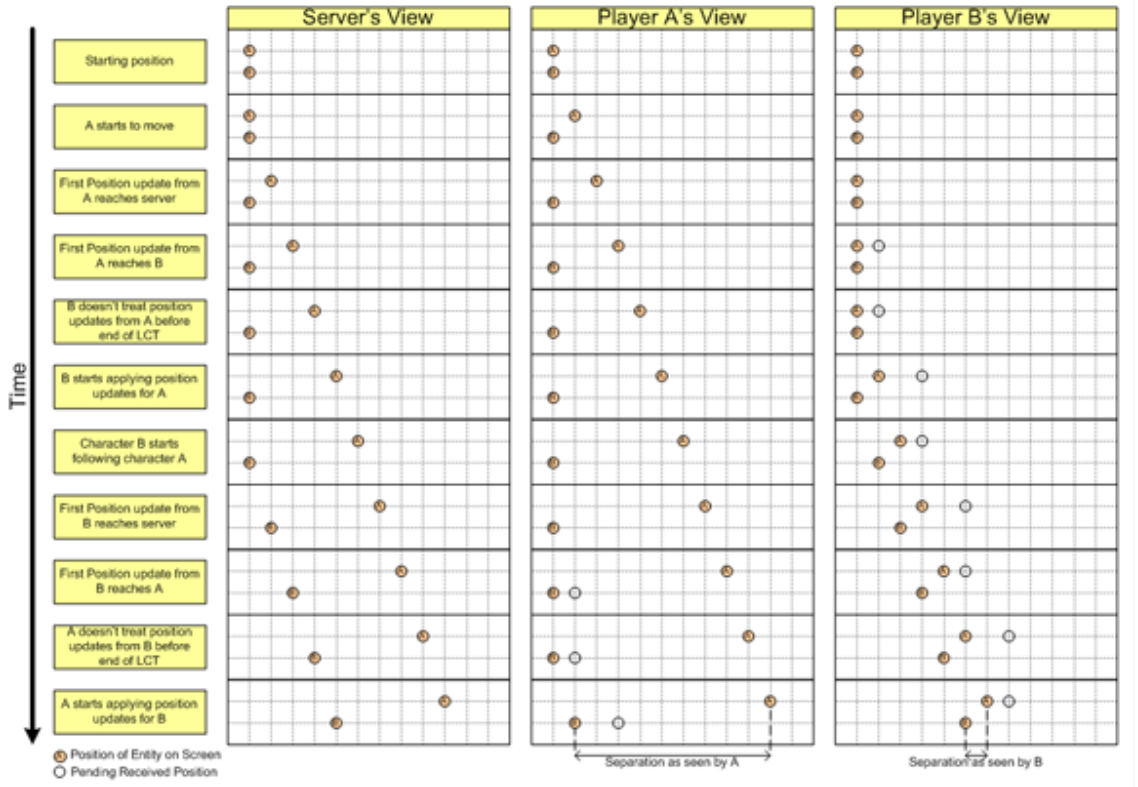

그림 3: 렉 보상 시간과 보간법

만약 두 명의 플레이어가 자신의 화면의 양측 끝에 서있다고 한다면(그림 3), 이들은 시간이 지남에 따라 화면이 전환되는 것을 볼 수 있다. 예를 들어 두 명의 캐릭터가 동시에 달려가려고 하더라도 이들은 서로 상대가 언제나 자신보다 뒤쳐져있는 것으로 느껴질 것이다. 왜냐하면 자신이 조종하는 캐릭터는 시간에 따라 전환되지 않기 때문이다.(이것은 정말 끔찍한 조작감을 자랑하게 될 것임이 틀림없다!)

이것은 현재의 시공간 내에서 표시되는 오브젝트(플레이어의 아바타)와 지난 시공간에서 표시되는 오브젝트(원격 캐릭터, AI 엔티티) 사이의 상호작용에 문제를 야기시킨다. 이에 대한 해결책은 플레이어 아바타의 거리에 맞춰 LCT 를 변화시키는 것이다. 이러한 구상은 동시적 지각(temporal perception) 또는 표현 시간(presentation time)이라 불리며, 개별적 지각 필터(local perception filters)라고 부르는 경우도 있다. 이러한 개념은 하늘에 보이는 별이 멀면 멀수록 빛이 우리에게 다가오는 시간이 오래 걸리는 원리와 흡사하다. [Singhal-Zyda]

그렇지만 만약 플레이어 아바타 주변에 좀 더 정확한 움직임을 구현하고 싶다면 관측자에게 가장 가까운 엔티티가 멀리 있는 엔티티보다 갱신에서 높은 우선순위를 가져야 할 것이다. 이것의 목표는 오차 범위를 최소화하는 것으로 최전면의 엔티티(근접 공격을 하는 적 등)는 1 미터 이하의 위치 오차가 난다고 하더라도 위치가 전혀 잘못 잡힌 것이 되어 치명적인 영향을 미칠 수 있다. 그렇지만 먼 곳에 위치한 엔티티는 위치 오차가 몇 미터 가량 난다고 하더라도 그것은 오직 몇 픽셀 정도의 오차로 밖에 보이지 않으므로 큰 문제가 되지 않는다. 원거리의 엔티티에 대해서는 낮은 주기로 갱신이 이루어지고, 요구 LCT 는 더욱 높다. 유연성이 있는 LCT 는 근처 캐릭터에 대한 렉은 최소화시킴과 동시에 배경에 있는 캐릭터들의 어색한 움직임도 피할 수 있다.

이 말은 즉 눈에 가시 엔티티의 갱신 주기를 조절할 수 있어야 한다는 것이다.

시간 동기화

여러 대의 기계 사이에서 시간을 유지하는 것은 동기화 메커니즘이 필요하다. 일반적인 동기화 방식은 클라이언트 기계의 시각과 기준 시각(reference time)을 비교해 기준 시각에 연관된 타임스탬프를 보내는 것이다. 그렇지만 우리는 많은 소비자 PC 가 서로 다른 속도의 내부 시계를 지니고 있는 바람에 동기화가 제대로 이루어지지 않게 된다는 것을 깨닫게 되었다.

게다가, 우리측의 서버는 안내 서비스에서 보내지는 “ticks”에 기반을 둔 유연성 있는 시간 체계를 채택하고 있었기 때문에 현재 게임 사이클을 모든 서버 어플리케이션에서 유지할 수 있는 속도로 증가시킬 수 있었다. 만약 서비스가 갑작스럽게 작업량이 늘어나게 되면 모든 서비스는 이것을 기다리게 되며, 정체 현상의 반복을 피할 수 있게 된다. 클라이언트 시간 동기화는 두 개의 수신 메시지의 평균에 기반으로 두고 진행되며 서버가 정기적으로 각 클라이언트에 메시지를 보낸다는 것을 가정하고 있다.

4. 갱신 주기 조절 - 우선순위 설정

모든 사람이 모뎀과 대역폭이 몇 년 이내에 크게 증가할 것이라고 믿긴 했지만, 낮은 전송률을 유지하는 것은 서버 대역폭을 낮춰서 비용을 절약할 수 있으므로 이점을 지닌다. 그렇지만 수천의 플레이어를 하나의 서버에서 돌리는 것(MMOG 의 에센스라 할 수 있는 것)은 많은 인터넷 접속을 필요로 할 것임이 명확하고, 당연히 필요한 비용 역시 높다.

많은 3D 물체가 있는 화면에서 위치의 변화를 13kbit/s 로 전송하는 것(최종적으로 결정된 한계였다.)은 다음과 같은 문제를 야기시킨다.

- 어떤 갱신사항이 우선순위를 지니는가?
- CPU 효율(CPU-efficiency)의 필요는?

대여섯 종류의 알고리즘을 시도해보았었다. 이것들의 원리는 모두 동일했다. 특정 관측자에게 동시에 다음과 같은 상황이 발생한다.

1. 어떤 엔티티가 관측자의 주변에 있는 것인지를 판단하고 목록의 변화(“비전(vision)” 이라 불리었다)를 클라이언트와 공유하라.
2. 이 리스트에 있는 각 엔티티에 거리에 따른 우선순위를 부여하라.
3. 우선순위가 높은 것과 낮은 것의 순위로 엔티티를 반복 적용하고 이들의 상태를 클라이언트에 의해 알려진 상태의 복제본과 비교해보라. 큰 차이가 발견될 경우 갱신 기록을 송신 버퍼(send buffer)에 추가하라. 전송 버퍼의 크기가 대역폭의 한계에 도달하면 멈춰라.

가시(visible) 엔티티의 목록 계산하기

*라이즘*에는 두 개의 중심 알고리즘이 사용된다.

하나는 “본토” *라이즘*에 사용된다. 공간은 격자(grid)로 나뉘어져 있으며, 가시 엔티티의 목록은 현재 구역과 그 주변에 나선형을 그리며 배치된 구역으로부터의 엔티티로 구성되어있다. 정해진 숫자만큼의 가시 엔티티가 찰 때까지 이것이 반복된다

본토보다 더 작은 지역인 *라이즘* 링(Ring)은 새로운 알고리즘이 사용된다. 엔티티에서 그룹의 중심력까지의 거리에 기반해 동적 그룹이 형성되는 방식으로 만약 엔티티가 그룹으로부터 너무 멀리 이동하면 그룹은 2 개로 나누어져서 새로운 그룹이 형성되게 된다.

두 사례에서 결과는 동일하다. 클라이언트마다 게임 엔티티와 가시 엔티티 사이의 관계가 존재한다. 이러한 알고리즘을 상세히 설명하는 것은 이 글의 주제를 넘어가는 것이긴 하지만 지금부터 이러한 연관 관계가 언제 어떻게 그 속성이 클라이언트로 전달되는지에 대해 알아보도록 하겠다.

우선순위: 정보의 적합성

우리가 시도한 알고리즘은 다음과 같은 접근 방식으로 제작되었다. 속성의 갱신 주기를 "픽셀의 수"와 연관시켜서 최종적으로는 뷰어(viewer)의 화면 속성 변화에 영향을 받게 하는 것이다. 실제로는 최초의 예산기반(budget-based) 알고리즘이 각 속성을 담당하도록 제작되었었다. 각 튜플(뷰어 클라이언트, 가시 엔티티, 속성)은 우선순위가 배정되고 "선반 & 양동이(shelves & buckets)"를 나타내는 데이터 구조는 클라이언트에 갱신 정보를 전달할 때 읽혀졌다. 이런 우선순위는 속성 변화 이벤트를 올바른 양동이(bucket)에 적용하는 것을 돕는다. 이는 다음과 같은 기준을 형성하였다.

- 관측자와 엔티티 사이의 거리(성능 향상을 위해 "*맨하탄* 거리(*manhattan* distance)" 근사값이 사용되었다.)
- 클라이언트와 실제 수치 사이의 차이의 규모

위치와 관련해, 우리는 엔티티가 긴 경로를 이동하였을 때 차이값이 낮다면 발생하는 문제에 직면했다. 예를 들어 벽 주변을 걷을 때라면 시작 위치와 종료 위치가 매우 근접하게 된다. 이 경우 변화가 "중요하다." 우리는 위치의 변화를 비교하는 대신 마일리지(mileage)라 불리는 "움직임의 양(quantity of movement)"을 비교하였으며, 우리는 매번 엔티티의 움직임이 발견될 때마다 그것을 저장해나가야 했다.

다른 속성도 고려되었다. 예를 들어 만약 가시 엔티티가 갑작스럽게 머리 위에 나타나 아무런 움직임을 보이지 않는다면 이 엔티티는 높은 갱신 우선순위를 지니게 된다.

이러한 알고리즘은 우리의 목표였던 25 x 250 x 1000 페어에는 CPU 를 너무 과도하게 요구하였다.(각 프론트엔드 서비스마다 1000 개의 클라이언트가 배정되며 각각은 250 개의 엔티티와 25 개의 동적 속성을 출력한다.) 따라서 다른 알고리즘이 개발되었다. 이것은 다음과 같은 기준을 따라 페어(뷰어와 가시 엔티티)의 스코어(score)를 계산했다.

- 관측자와 엔티티 사이의 거리(성능 향상을 위해 "*맨하탄 거리(manhattan* distance)" 근사값이 사용되었다.)

엔티티의 스코어는 뷰어까지의 거리와는 반비례해 균형적으로 성장하였으며, 뷰어 클라이언트에게 수치를 전달할 때에는 리셋이 이루어졌다. 특정 가시 엔티티를 처리할 때 어떤 속성을 뷰어에게 전달시켜야 하는지에 관련된 처리도 필요했다.

전송해야 할 속성 중재하기

가시 엔티티를 우선순위의 내림차순에 따라 배열하는 것을 반복할 때 우리는 속성의 현재 값과 지난 번에 보내진 저장된 값을 비교하게 된다. 만약 이 두 수치가 맞지 않는다면 그것을 보내질 예정의 갱신 기록에 포함시킨다. 앞선 글에 언급된 마일리지 트릭은 위치 조정을 위해 유지한다. 여러 번 큰 수치를 보내는 이러한 비교는 중요한 부분이므로 이것을 최적화시키기 위해 여러 가지 트릭을 이용하였다. 나중에 우리는 엔티티 종류에 연관된 속성만을 비교함으로써 전체 과정을 최적화시켰다. 예를 들어 우리는 *라이즘*의 *인텔리전트 플랜트(intelligent plant)*가 고정 엔티티임을 알고 있기 때문에 이것의 위치를 비교할 필요는 없다.

정량적 평가(Quantitative Evaluation)

전략과 그것의 시행을 평가하는 것은 쉬운 일이 아니다. 왜냐하면 고정된 대역폭 사용에 기반해 이것들은 클라이언트 내에서 주관적 움직임과 애니메이션 크레디빌리티 레벨(animation credibility level)로 나타나기 때문이다. 게다가 다른 이동 화면을 여러 번 비교하는 것은 정량적 데이터 없이는 비교가 불가능하다. 이것이 바로 우리가 프론트엔드 서비스 내에 측정(measurement)을 만들고 이러한 과정을 간소화시키기 위한 그래프를 만든 이유이다.

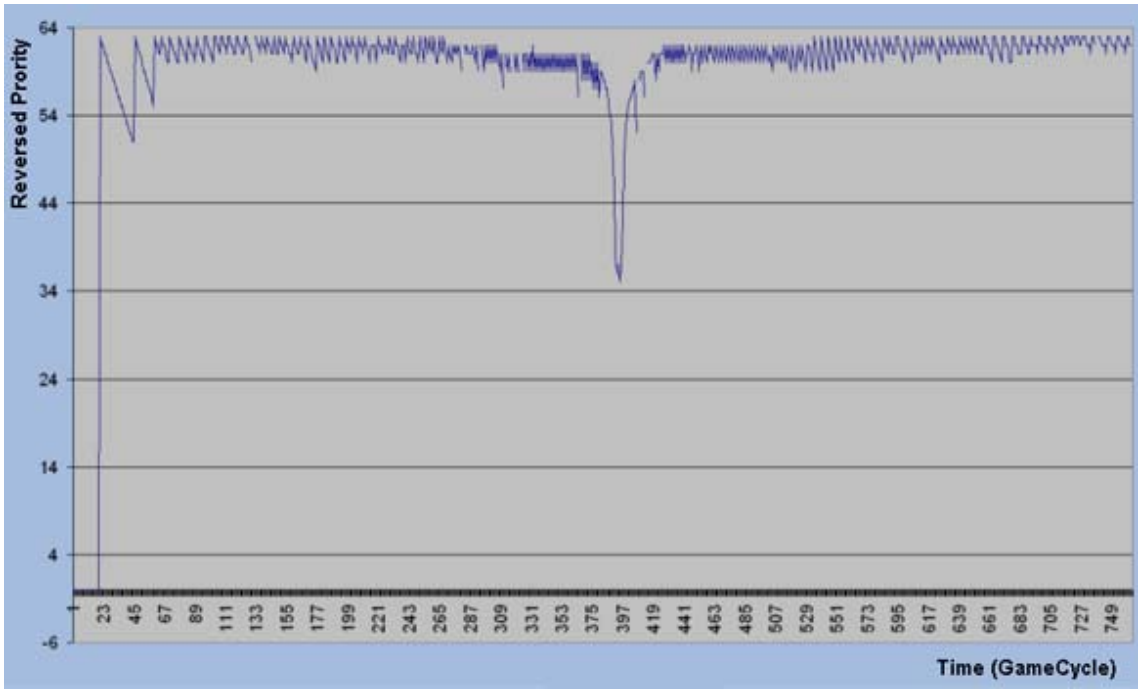

그림 4: 시간의 흐름과 엔티티의 위치에 따른 우선순위

그림 4 는 다수의 엔티티가 일정한 속도로 움직일 때 변화하는 위치의 우선순위를 측정한 예제로, 뷰어는 이것의 따라 이동한다. 이 그래프에서 우선순위가 높을수록 y 값은 낮고 갱신이 이루어질 가능성은 높아진다. 가장 일반적인 패턴은 톱날 형태이다.(감소하는 직선이 수직선으로 잘린다.) 사실 보여지는 엔티티(watched entity)는 움직이고 있으며 우선순위는 갱신이 이루어질 때까지 계속 증가하게 된다.

그래프는 흥미로운 현상 두 개를 보이고 있다. 하나는 바로 첫 번째 두 패턴이 뒤의 패턴들보다 더욱 크다는 것이다. 이것은 서버가 처음에는 모든 첫 속성을 클라이언트에 보내야 하기 때문이다.

그렇지만 대역폭에는 동일한 한계가 존재하기 때문에 이러한 현상이 발생한다. 보여지는 엔티티는 높은 우선순위를 지닌 더 많은 경쟁 엔티티를 지니기 때문에 전송은 평균보다 더 높은 우선순위(낮은 y 값)에 한해 이루어지게 된다. 이러한 경쟁과는 별개로, 최초의 갱신은 커맨드(비쥬얼 속성과는 별개)를 서버에서 클라이언트로 보내기 위해 사용되는 임펄젼(impulsion)이라는 다른 네트워크 메시지와 함께 대역폭을 위해 경쟁하게 된다.

다른 가시 큐는 그래프의 중간 부분에서 확인할 수 있다. 이것은 뷰어가 그룹 중앙의 엔티티를 지나갔을 때 발생한다. 뷰어와 가시 엔티티의 거리가 줄어듦에 따라 우선순위와 갱신 주기는 증가한다. 대부분의 경우 보여지는 엔티티는 뷰어와 그 주변의 엔티티가 지니는 거리와 흡사한 정도의 거리에 떨어져있으나, 곡선의 중간에 있는 짧은 시간 프레임 동안에는 다른 것들보다 뷰어에게 더 가깝게 다가가게 된다.

정성적 평가(Qualitative Evaluation)

Snowballs(NeL 의 기술 데모)에 기반을 둔 가벼운 클라이언트는 풀 *라이즘* 클라이언트를 완성하기 전 정성적 평가 결과를 가시적으로 확인하기 위해 개발되었다. 이 프로토타입은 클라이언트 측에 위치하였으며, 주기 조절을 위해 도입된 프론트엔드 코드를 위한 테스트 어플리케이션을 제공하였다. 엔티티의 위치 갱신 주기는 서로 다른 색깔로 표시되었다.

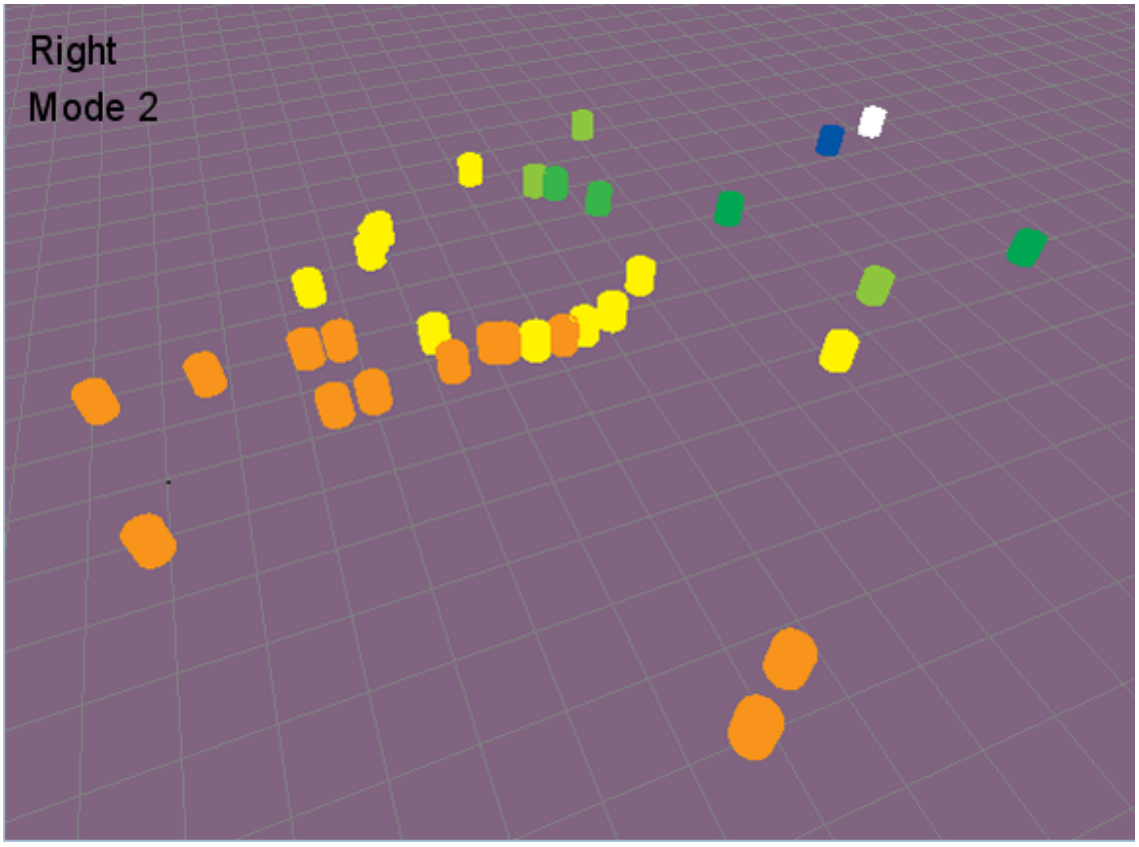

그림 5: 갱신 주기의 가시적 확인

위의 스크린샷(그림 5)는 근거리의 뷰어(하얀색)에 가까운 엔티티(파란색과 녹색)가 먼 곳에 있는 엔티티(노란색과 오렌지색)보다 높은 갱신 주기를 지니는 것을 볼 수 있다.

최적화

게임 사이클에서 이루어지는 작업을 수를 줄이기 위해서 계산을 여러 게임 사이클로 나누는 시스템이 추가되었다. 그 결과로 가시성 페어의 증분 서브셋(subset)만이 한 번에 우선순위를 지니게 되었다.

또 다른 최적화는 멀티프로세서 머신(나중에는 하이퍼 쓰레드 멀티프로세서를 이용했다.)의 이점을 취하는 것으로, 계산 파트와 전송 파트는 여러 쓰레드를 이용하는 파이프라인 방식을 사용하였다.

저레벨 최적화 역시 이루어졌었다. C++ 컴파일러에서 생성된 어셈블리 코드를 출력하는 것을 통해 우리는 데이터 구조를 재가공할 수 있었고

이를 통해 프로세서는 접근 시 최소한의 오버헤드와 캐시 미스를 지니게 되었다.

결론

결국 우리가 예로 든 프론트엔드 서비스들은 동적 속성을 필터하고 우선순위를 부여하고 이것을 대략 수천의 클라이언트에 각각 전송하는 것이 가능하게 되었다. 이 시스템의 측정가능성(scalability)은 하나의 심리스 환경(샤드)에서 지원해야 하는 플레이어의 수에 따라 프론트엔드 서비스의 수를 증가시킬 수 있게 해주었다. 우리는 게임 샤드마다 하나의 멀티프로세서 서버와 이것과 동일한 여러 개의 서버에 프론트엔드 서비스 다수를 병렬화해 구동하는 것이 가능해졌다.

물론 이 시스템이 어떤 속성 변화 이벤트가 클라이언트로 보내질 것인지를 보증하진 않는다. 일부 속성 변화를 동기화시키기 위해서는 타임스탬프와 같은 다른 정보들이 이용된다.

5. 데이터 비교

소규모로 나뉘어진 존(zone)에 기반을 두는 온라인 게임은 적은 범위가 필요한 위치 좌표 로컬(position coordinates local)을 존에 사용할 수 있다. 하지만 거대한 심리스 환경은 이론적으로 넓은 범위의 좌표를 처리해야 한다. 우리는 서버 측에 밀리미터 단위의 정확성을 지닌 32-bit 정수 위치를 지니고 있었다. 접근 방식은 절대 위치(absolute position)와 최신(latest) 절대 위치에 연관된 “작은(smaller)” 위치를 번갈아 가며 접근하는 것이 되어야 한다. 하지만 이러한 기술은 그 결과로 제한을 지니게 되므로 우리의 최선(best-effort)(신뢰도 낮은)의 경량(lightweight) 네트워크 프로토콜에는 쉽게 적용하기 힘들었다.

더욱 적합한 접근은 클라이언트가 오직 뷰어를 감싸고 있는 평방킬로미터 내의 주변 엔티티만을 출력한다는 가정에 기반을 둔 접근 방식이었다. (그림 6) 그리고 여기에 가상 로컬 기준(virtual local basis)이 사용된다. 뷰어에게 절대 기준에서 상대적인 기준으로

위치를 전송하는 것 대신에 “잘려진 절대 좌표(truncated absolute coordinate)” 알고리즘을 사용해 더욱 CPU 효율적인 방식이 구성되는 것이다.

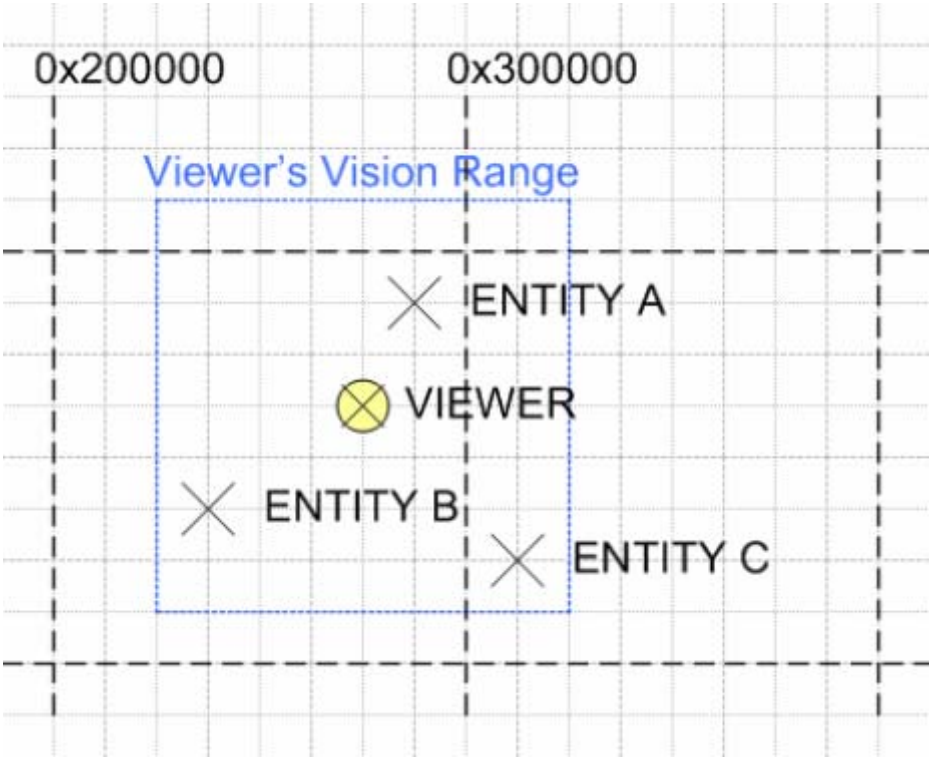

그림 6: 로컬 포지션 윈도우(Local Position Window) 예제

서버는 쇼트 엔티티(short entity) 좌표를 전달하며, 더 많은 대역폭을 보존하기 위해 1024 해상도의 윈도우에서 좌표를 16 으로 나누어 정확도를 16 밀리미터로 낮춘다.(예: 4bit 로 변환) 뷰어의 위치에서 레퍼런스(reference)는 필요하지 않다.

- uint16 ec_short = (uint16)(ec_full >> 4);

엔티티의 전체 위치(full position)을 재구성하기 위해서는 클라이언트는 아래의 입력을 필요로 한다.

- 전체 뷰어 좌표(Full viewer coordinates) (uint32[1] vc_full)
- 쇼트 엔티티 좌표(Short entity coordinates) (uint16 ec_short)

클라이언트는 쇼트 뷰어 좌표를 계산하고, 전달된 수치에서 쇼트 엔티티 델타 좌표는 최종적으로 감쇠된 뷰어 좌표(damped viewer

coordinate)(뷰어가 이동할 때 엔티티가 깜빡이는 것을 방지)와 풀 델타(full delta)(부호 연장(sign extension)으로부터 획득)로부터 풀 엔티티(full entity) 좌표를 추론해낸다.

- uint32 vc_full_damped = vc_full & ~0xF;
- uint16 vc_short = (uint16)(vc_full >> 4);
- sint16 delta_short = (sint16)(ec_short - vc_short);

- sint32 delta_full = ((sint32)delta_short) << 4;
- uint32 ec_full = (uint32)(vc_full_damped + delta_full);

그림 6의 예제:

Full X Coordinates:	Shrort Coordinates:	Short Delta:	Full Delta:
뷰어: 0x2C0000	뷰어: 0xC000	엔티티 A: 0x2000	엔티티 A: 0x00020000
엔티티 A: 0x2E0000	엔티티 A: 0xE000	엔티티 B: 0xA000	엔티티 B: 0xFFFA0000
엔티티 B: 0x260000	엔티티 B: 0x6000	엔티티 C: 0x6000	엔티티 C: 0x00060000
엔티티 C: 0x320000	엔티티 C: 0x2000		

만약 그라운드 엔티티를 다룰 경우에는 레이어 인덱스를 참조해 z 값(높이)가 더 줄어들 수도 있다. 클라이언트 소프트웨어는 2D 값에 연동해 3D 지형 구조에 효율적으로 접근하는 것이 가능해진다.

1 NeL 라이브러리에서 볼 수 있는 기본 타입들은 uint32 는 32-bit unsigned int이고, sint16 은 16-bit signed int와 같은 형식을 띄고 있다.

6. 시간 예측(Time Prediction)을 통한 지터값 감소

이제 우리 클라이언트의 애니메이션 시스템은 주어진 시간에 속성(특히 위치) 갱신이 이루어지며 특정 엔티티는 다른 엔티티보다 갱신 주기가 더 짧기도 하다. 엔티티의 실제 애니메이션은 지연으로 시작되는 수신 위치 갱신과 이동 속도를 이용해 클라이언트는 엔티티가 갱신된 위치에 출력되기 전에 미리 최소 하나의 위치값을 수신하게 된다.

이것은 지속적인 갱신 주기를 지닌다면 쉽게 해결될 수 있다. 그렇지만 특정 엔티티의 부드러운 움직임을 위해서는 이동 애니메이션의 시간 조절이 피드 데이터의 주기 변화에 대응해야 한다. 만약 엔티티가 다음 위치값을 수신하지 않았다면 이동을 멈추고 수신이 완료된 후에 다음 위치로 갑작스럽게 이동해가게 될 것이다. 이것이 바로 우리가 피하고자 하는 것이다.

그러므로 우리는 주기 변화를 예상하고 이동 속도를 미리 조절할 필요가 있다. 우리는 그렇게 하기 위한 다양한 알고리즘들을 비교해 보았다.

위치값 갱신 주기의 통계적 예측

다양한 이동 방식을 시뮬레이션하기 위해 테스트 벤치가 개발되었으며, 갱신 주기의 측정은 시뮬레이션 된 클라이언트에서 이루어졌다. 예측된 인터벌(interval)을 이용해 이동속도를 조절하기 위해 최신 수신 데이터를 기반으로 예측 공식들의 여러 개 계산되었다. 여기서 인터벌은 대부분의 경우 실제 갱신 인터벌보다 미세하게 높은 값을 보여줄 필요가 있었다. 우리의 초기 스코어 시도는 성공적이었으며, 우리는 이론적 작업에서 이득을 얻을 수 있도록 하기 위해 연구 논문들을 살펴보았다.

비슷한 시도를 하던 과정에서 Voice-over-IP 기술을 발견할 수 있었다. 이 경우에는 사운드를 목소리 끊김 없이 [Moon-Kurose-Towsley] 가능한 최소의 렉으로 전달하는 것이었다. 우리는 연구 논문에서 [Kansal-Kandikar] 우리의 최초 시도보다 효율성을 높여줄 수 있는 알고리즘을 채택하였다. (그림 7)

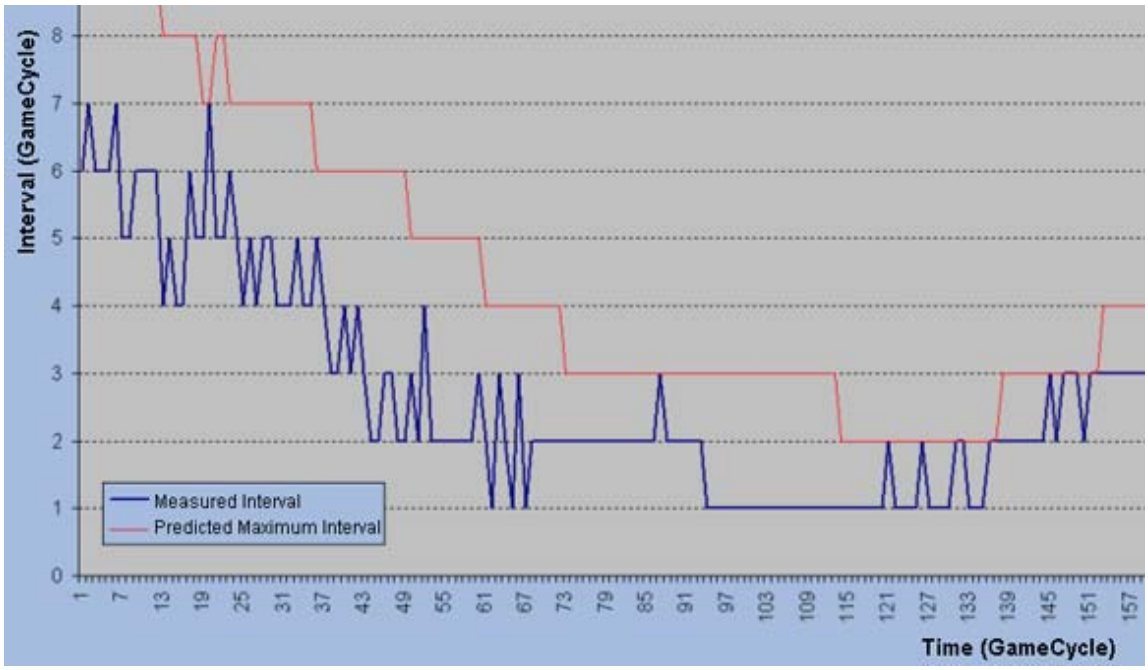

그림 7: 측정된 위치값 갱신 간의 인터벌(파란색)
그리고 시간에 따른 상대적인 예측 최대 인터벌(붉은색)
만약 예측된 곡선이 측정된 곡선과 겹치게 되면 이동에 끊김이
발생하게 된다

그렇지만 이 기술은 엔티티가 이동을 멈춰서 위치값 업데이트 갱신을 멈출 경우를 다루기 위해 개선을 거칠 필요가 있었다.(우선순위가 낮아져서 그러는 것이 아니라)

7. 결론

새롭게 개발된 기술은 움직이는 엔티티의 부드러운 모습을 보이는 것에 성공적이었다. Nevrax 팀의 멤버들은 수백의 애니메이션된 엔티티들이 움직이는 테스트를 살펴봤고, 이것이 선보이는 대규모 전투 장면에 열광적인 반응을 보였었다.

나중에 속성 전송(property propagation) 시스템이 실시간 군대 습격 배치(Realtime Army Invasion Deployment)(RAID) 엔진에 포함되게 되었는데, 이 엔진은 NPC의 행동을 다루는 AI 서비스부터 시작해 엔티티를 애니메이션하는 3D 클라이언트 엔진까지를 다루는 것이었다. 이것은 수천 마리의 키틴(Kitin)(꽃게 같이 생긴 생명체들)과 수백 명의 플레이어가 싸우는 대규모 이벤트를 처리하는데 사용되었다. 이러한 장면은 마치 영화 *스타쉽 트루퍼스(Starship Troopers)*를

연상시키는 것이었고, 이 실제 게임 영상에서도 이것을 살펴볼 수 있다.

그림 8: 키틴의 습격을 담은 게임 스크린샷

8. 참고

- [Aronson] Aronson, J., "Dead Reckoning: Latency Hiding for Networked Games", 1997
http://www.gamasutra.com/features/19970919/aronson_01.htm

- [Moon-Kurose-Towsley] Moon, S.B., Kurose, J., Towsley, D. "Packet audio playout delay adjustment: performance bounds and algorithms", 1998
http://an.kaist.ac.kr/~sbmoon/paper/intl-journal/1998-acm-multimedia-voip.pdf

- [Kansal-Karandikar] Kansal, A. Karandikar, A. "Jitter-free Audio Playout over Best Effort Packet Networks". ATM Forum International Symposium, New Delhi, 2001
http://citeseer.ist.psu.edu/557994.html

- [Bernier] Bernier, Y. B., “Latency Compensating Methods in Client/Server In-Game Protocol Design and Optimization”, Proceedings of the Game Developer Conference, 2001
http://www.resourcecode.de/stuff/clientsideprediction.pdf

- [Singhal-Zyda] Networked Virtual Environments: Design and Implementation, Singhal S., Zyda M., Addison-Wesley Professional, 1999, ISBN 0201325578.

(링크는 2007 년 5 월 연결 확인)

(역주: 2007 년 10 월 2 일 연결 확인)