| **비단길(Silk Road)** | |
| --- | --- |
| **16주차** | **2024.04.07 (일) ~ 2024.04.13 (토)** |

**■ 주간 회의**

**▶ 3월 작업물 병합**

* assimp로 변경한 모델로드 방식과 물리엔진 부분을 일단 병합 후 테스트를 마치고, 후에 네트워크 부분을 병합했다.
* assimp 헤더의 위치 때문에 빌드가 되지 않는 문제를 제외하고 문제 없이 진행할 수 있었다.
* 프로그램 또한 우리가 예상했던대로 정상적으로 동작했다.
* 앞으로 남은 일정동안 현재 커밋을 기반으로 인게임 내용을 제작할 예정이며, 서버는 새로 작업중인 부분이 완료되면 다시 병합을 시도할 예정이다.

**■ 다음 주 세부 계획**

**▶ [권순원]**

* 서버 방식 변경 계속 작업
* 다음 주 중으로 서버 완성 후 메인에 병합하기

**▶ [박준영]**

* 인게임 콘텐츠 개발
* ai 개발

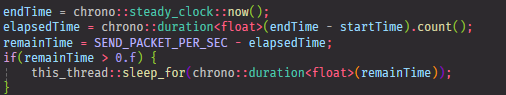
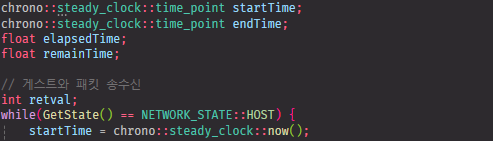
**▶ [엄장헌]**

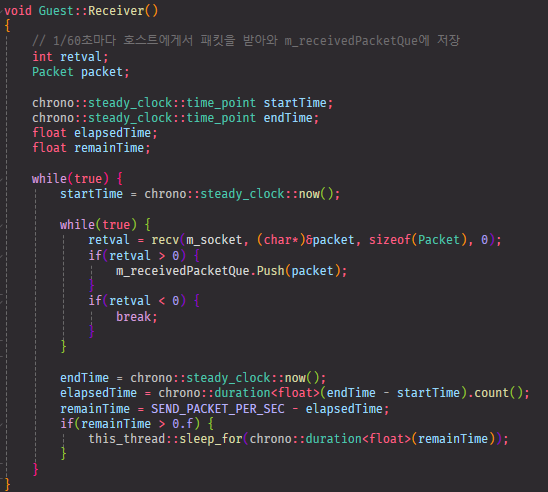
* 물리엔진에서 충돌시 회전량과 마찰력 다시 구현

**■ 작업 일지**

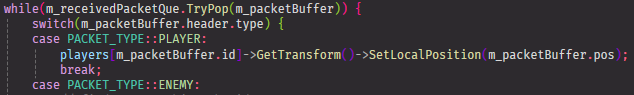
**▶ [권순원]**

1/60초를 계산하는 부분을 개선했다. 지난 주는 단순히 지난 시간을 더해 확인을 하고 스레드를 계속 동작시키는 것으로 만들었는데, 이러면 스레드가 크진 않지만 CPU의 자원을 사용한다고 생각이 되었다. 남은 시간을 계산해 스레드를 정지시키는 방식으로 변경하였다.





게스트에 Receiver함수를 추가했다. 지난번에 만든 서버에서는 프레임마다 recv를 호출해 패킷을 적용하였는데, 지난 주에 결정한 방식으로 서버를 변경하면서 이 부분도 변경할 필요를 느꼈다.   
 별도의 스레드로 recv를 호출해 패킷을 받고 이를 클라이언트에서 업데이트에 사용할 LockQueue에 Push를 하는 역할을 한다.

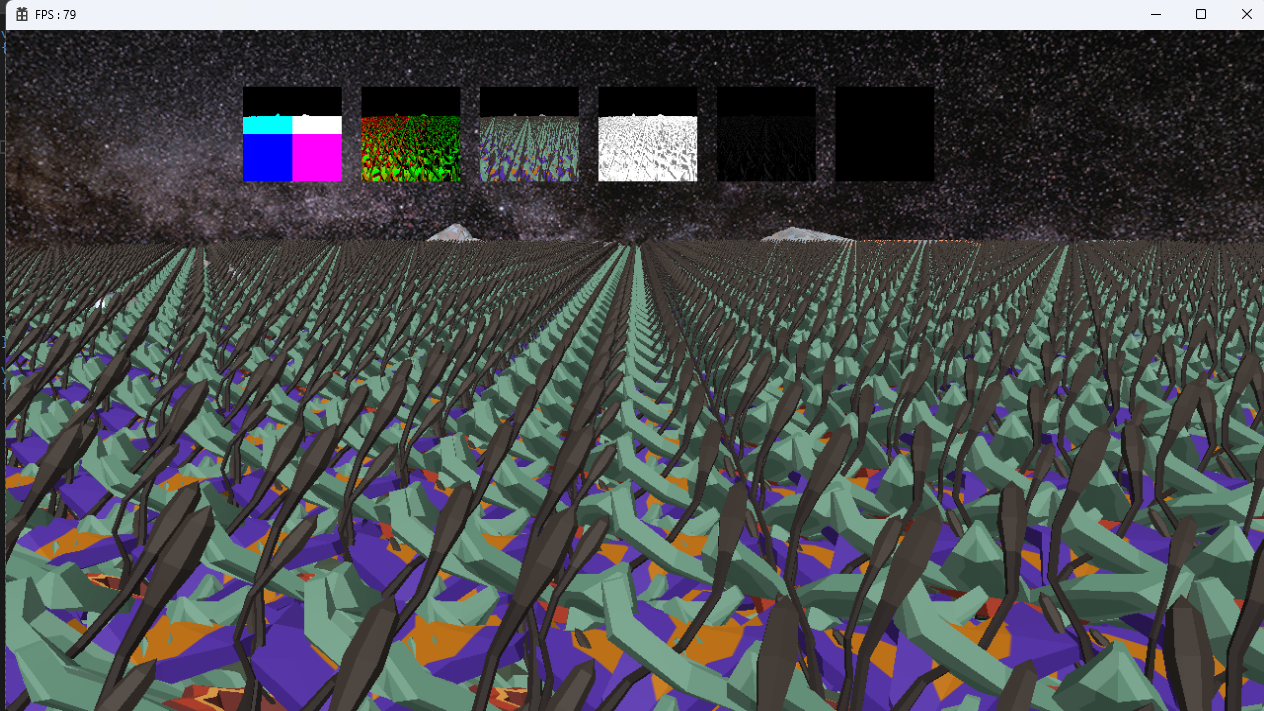


Receiver함수를 추가함으로 클라이언트에 적용할 패킷을 모아둔 LockQueue를 제작했으므로, 기존에 Update에 함수에서도 해당 큐를 사용하도록 변경하였다.

**▶ [박준영]**

이전까지 100개 이상의 다량의 오브젝트 메시를 한 화면에 표현 시 파이프라인 부분에서 문제가 생기던 부분을 수정하였다.

상수버퍼와 디스크립터 힙의 생성인자를 변경하여 대량의 오브젝트를 한 화면에 표시하는 것을 성공 후, 개수에 따른 프레임 변화율을 확인해보았다.



예시로 사용된 꽃 오브젝트를 최대 6000개 가량 까지 배치 할 수 있었고, 5000개 정도까지 초당 프레임 60 이상을 안정적으로 표시할 수 있었다.

물리엔진이 적용되는 객체는 저정도 수준까지 늘릴 수는 없겠지만, 프러스텀 컬링과 안개효과를 적용하고 순수하게 장식용으로 사용되는 객체에 한해 리지드 바디 컴포넌트를 제거함으로써 보다 풍성한 오브젝트들을 안정적인 프레임 레이트와 함께 제공 가능 할 것으로 보인다.

전 주부터 진행하던 모델 로드 기능 분류를 완료하였다.

assimp를 통해 로드할 메시와 fbxsdk를 통해 로드해야 하는 애니메이션 객체를 animclip의 로드 여부로 분류하여 각기 다른 함수를 통해 로드하도록 하였다.

****

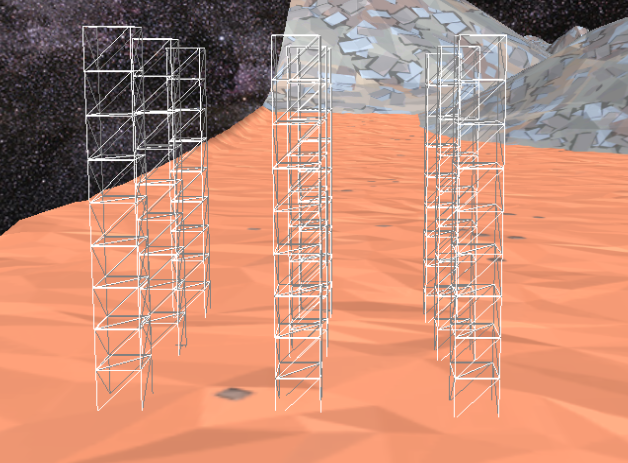
유니티를 통해 불러오는 일부 모델의 프리팹은 계층적 구조를 갖고있다.

예: 자동차의 경우 자동차 본체 -> 4개의 바퀴, 4개의 문짝

만약 이러한 계층 구조를 통해 자동차의 문이 열리거나 바퀴가 돌아가는 기믹을 구현할 예정이라면 계층구조를 유지하여야 하지만, 우리게임에서 인간형 모델의 경우 별도의 애니메이션 구조를 사용하며, 일부 터렛이나 비인간형 보스 몬스터의 경우 아예 여러 객체를 SetParent를 통해 게임오브젝트 단위에서 계층적 연결을 할 예정이기에 계층적 메시 구조를 하나의 메시로 통합하는 코드를 unity를 통해 구현하였다.

이 코드를 통해 unity 3d editor 상에서 원하는 객체를 골라 툴을 적용하면 하나의 하위의 게임 오브젝트들의 메시를 통합하여 동일한 로컬 좌표계의 단일 메시로 추출된다.

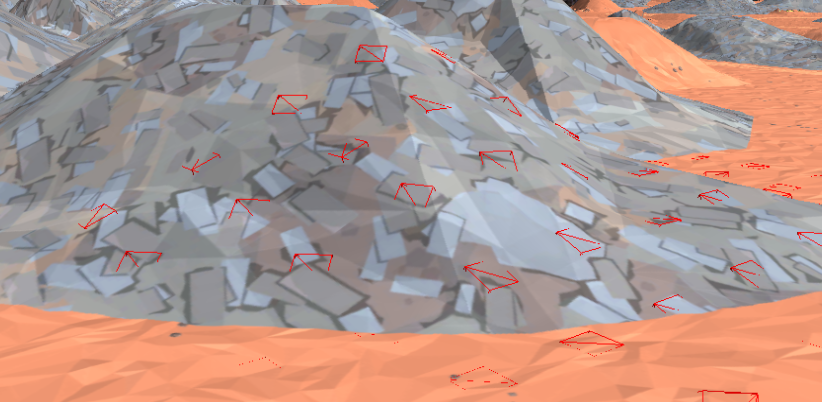
**▶ [엄장헌]**

물리엔진을 전체적으로 수정하였다. 이전에 사용하던 방식은 DirectXCollision.h에서 제공하는 intersects 함수를 이용해서 충돌체크를 하였다. 이 경우 문제가 있었는데, 객체가 겹쳐진 크기나, 방향을 알려주지 않고 닿았다는 정보만 bool로 리턴되었다. 이 때 그냥 속도와 질량을 이용해서 충돌처리를 해줬었다. 하지만 이처럼 처리하면 계속 닿아있는 강체, 예를들어 쌓여있는 경우, 연속적으로 계속 밀고있는 경우 등에서 정확한 계산이 적용되지 않아 부들거리거나 떨리는 현상이 있었다. 따라서 직접 intersects 함수를 만들었다. SAT를 이용해 충돌처리를 하는 과정안에서 최소의 크기로 겹쳐진 방향벡터를 찾고, 그 거리만큼 정확히 절반씩 이동 후에 힘을 가해주었다. 다만 static 객체와 충돌하는 경우는 움직이는 객체만 이동후 힘을 받는다. 이를 통해 안정적인 강체간 stacking을 할 수 있다.

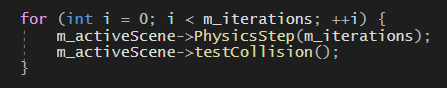
또한 탄성계수를 적용해서 통통 튀는 듯한 재질과 단단한 느낌의 재질도 만들었다.

그 다음으로 굉장히 고민한 것이 Terrain과의 충돌처리이다. 이전의 높이만을 가지고 오는 방식으로는 산에 부딪히거나 바닥에 떨어졌을 때의 처리, 물체가 바닥을 구르거나, 경사가 급한 지역을 중력에 의해 올라가기 힘들게하는 등의 처리가 하드코딩으로 해결해야 된다. 따라서 위에서 만든 intersects를 사용하고 싶었다. 그렇다고 모든 터레인의 삼각형 부분을 객체로 만들고 충돌을 처리하면 너무 많은 자원이 들게 된다. 따라서 충돌 처리에서 터레인에 Collider가 있는 척을 하였다.

원래 터레인의 높이값만 가져와서 비교했는데, 그 대신 객체의 xz값으로 해당 터레인 위치에서 법선벡트를 바라보는 OBB와의 충돌을 처리하게 하였다. 이를 위해 터레인에서 해당 위치에서의 주변 높이들을 이용해 법선벡터를 얻어냈고 그것을 이용해 OBB의 Look 벡터가 법선벡터를 향하도록 회전할때 구해지는 8개의 점으로 충돌처리를 하였다. 결과적으로 잘 적용되었고, 경사면에서 법선벡터 방향으로 가해지는 힘과 중력에 의해 잘 오르지 못하고 떨어지게 된다.



위 사진은 모든 점에서의 콜라이더를 미리 만들어 놓은 것이 아니라 해당 점에선 저런 OBB가 있는척하여 충돌한다는 것을 보여주기 위함이다.

다음으로 만든것은 물리엔진의 정확도를 높이기 위한 iterations이다. 컴퓨터의 물리엔진에서 문제점은 한 프레임의 이동 변화가 이산적이라는 것이다. 즉 속도가 너무 빠르면 충돌이 진행되지 않고 지나가게 된다. 이를 위해 한 프레임에서의 물리연산을 iterations만큼 쪼개서 진행하는 것이다. 

이를 통해 물리 연산의 정확도를 배로 높일 수 있다. 다만 문제점은 그만큼 성능을 갉아 먹는 것이다. 그래도 대부분 랜더링에서 자원을 먹는 만큼 충돌연산에서의 자원은 상대적으로 조금 쓰이기에 iterations가 2~4 정도에서는 프레임 변화가 크게 없었다. 최종적인 결과에서 프레임이 많이 남는다면 더 높일 가능성은 있을것 같다.

이제 충돌에 따른 회전을 가하려고 했는데, 동시에 여러개의 점에서의 충돌을 갖는 경우 정확한 회전량을 구할 수가 없었다. 이를 위해 충돌점을 저장하고 한번에 계산하는 방식을 택해야 할 것 같다.

애니메이션 모델 리소스를 가공하였다. 기존에는 캐릭터마다 모델정보와 애니메이션을 각각담아 fbx로 저장했었다. 대신 사용할 캐릭터들의 정점정보와 뼈, uv등 정보들을 하나의 fbx에 전부 담았고, 애니메이션을 이곳에 함께 담아 한번 로드시 모든 정보를 가져올 수 있게 하였다.



방학동안 만든것을 갈아엎었다는 점에서 너무 마음 아팠지만, 그래도 휴강과 공휴일 덕분에 작업에 길게 집중할 수 있었다. 또한 결과적으로 충돌 처리의 퀄리티가 훨씬 높아져서 만족스러웠다.