1. PQCS를 이용해 Timer Thread의 bottleneck 완화 시도 – 2 (오류 수정)

이전 작업에서 몬스터의 PQCS 과정에서 오류가 있었다.

Scene::scene->MonsterObjects 컨테이너가 concurrency\_vector<Character\*>로 이루어져 있는데 이 안에 있는 객체를 복사해서 사용하려 했기 때문에 오류가 난 것으로 보인다.

해결 방안으로 Character\* 형의 주소를 key로 보내는 것이 아닌 컨테이너 위치 인덱스를 key로 보내 객체를 찾는 방법을 이용했다.

서버 에러가 나는 오류는 해결했지만 클라이언트에서 연결을 시도하면 서버에 접속되지 않고 클라이언트가 종료되는 새로운 오류가 나타났다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

PQCS만 추가했는데 클라이언트가 서버와 연결되지 않는다는 부분에서 예상했던 원인은 너무 많은 메시지가 쌓인 것을 문제점으로 추측했다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Timer Thread의 한 번 loop마다 MonsterObjects를 돌며 많은 메시지를 보내고 있다. 이 부분에 delay를 두어 일정한 시간마다 메시지를 보내기로 해봤다.

하지만 여전히 같은 문제가 발생했고 원인이 메시지가 많아 쌓인 게 아니라는 것을 알았다.

1. 미끄러지는 충돌 처리

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

이전 충돌처리에서 플레이어가 맵 오브젝트와 부딪히면 단순히 속력을 0으로 맞춰 움직이지 않고 박히는 듯한 효과가 있었는데 이런 부분이 만족스럽지 않아 좀 더 자연스럽게 바꿔보려 했다.

도표, 스크린샷, 직사각형, 디자인이(가) 표시된 사진

자동 생성된 설명

여기서 자연스럽게 부딪힌다는 것은 움직이는 물체가 다른 물체와 사선으로 부딪힐 때 빗면에 따라 이동 경로를 바꿔줄 수 있다는 의미이다.

전과 달라져야 할 점은 충돌하는 대상의 OBB에서 어떤 면과 충돌되었는지 파악해야 한다.

이제는 단순히 물체끼리 충돌되었다는 것만 확인하는 것이 아니라 어떤 면과 충돌되었는지 확인하고 그 면의 Normal을 이용하여 움직이는 방향을 바꿔야 한다.

즉, 이제는 OBB나 BoundingSphere 간의 충돌체크가 아니라 어떤 면과 부딪혔고 그 면의 Normal은 무엇인지를 알아내야 한다.

BoundingBox의 8개의 점을 알아내어 그 점들로 면을 만들어 충돌체크를 하려 했고, 다행히 DirectX의 BoundingBox 멤버 함수 중 GetCorners()으로 8개 점을 알아낼 수 있다는 것을 찾았다.

GetCorners() 정점 순서

텍스트, 스크린샷, 흑백, 블랙이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

도표, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명

GetCorners() 함수가 반환하는 8개의 정점이 어떤 순서대로 반환되는지 알고 있어야 삼각형을 만들어 낼 수 있기 때문에 순서를 알아내는 테스트를 했다. 그 결과 위 그림의 순서로 반환하는 것으로 알 수 있었다.

8개의 정점을 알아냈으니 이제는 그 점들로 면을 만들 차례다.

3개씩 짝지어 CW 순서로 하나의 삼각형을 구성하며 총 12개의 삼각형으로 충돌체크를 했다.

그렇게 준비한 삼각형 12개와 다음 움직임이 일어나기 전 속도를 갖고, DirectX에서 지원하는 Intersects() 함수에서 면과 벡터를 인자로 넣어 확인하는 방식으로 충돌 여부를 파악하여 어떤 삼각형과 충돌되었는지 알 수 있다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

코드 구현 내용

삼각형의 Normal값을 구하는 방법은 세 점으로 두 벡터를 만들고, 그 벡터를 CW 순서로 외적하면 된다.

하지만 일부 면에서 충돌이 제대로 적용되지 않았다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

* Else 문으로 분기하는 것이 문제였다. OBB가 얇을 때 if문 순서에 따라 플레이어를 바라보는 쪽의 면이 아닌 반대 면과 충돌이 먼저 되는 것으로 판정되어 실제 충돌되어야 할 면에는 충돌처리가 진행되지 않았다.
* 따라서 충돌될 면이 은면인지 먼저 파악하고 처리 여부를 진행한다. 그렇게 하면 자연스럽게 전 보다 적은 양의 면을 검사하기 때문에 성능도 향상될 것이다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

최종 정리된 코드

우선 하드코딩을 최대한 피하고 줄 수를 줄이기 위해 CornersIndex라는 이름으로 삼각형 순서를 적어뒀다. 그리고 각 삼각형의 Normal 먼저 구하고 속도와 내적으로 충돌처리를 할 것인가 파악한다.

그 후 충돌되었는지 Intersects()로 체크되면, 슬라이딩 벡터(S = P – n(P\*n))로 속도를 바꿔준다.

충돌처리가 제대로 적용돼서 이제 미끄러지듯이 부딪히며 물체 위에도 설 수 있다.

1. 플레이어 카메라 오류(이름 바꿔야함)

게임을 실행하고 움직이다 보면 간혹 플레이어 위치가 이상한 곳으로 이동되거나 카메라에 보이는 그림이 정상적이지 않은 경우가 일어난다.

스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

카메라(플레이어 위치) 오류 문제가 나는 모습

충돌처리 때 문제가 자주 발생해서 처음엔 충돌처리 문제라 생각했으나 충돌처리를 꺼도 점프를 반복하다 보면 낮은 확률로 발생한다.

바로 생각나는 원인은 다음과 같다.

1. 플레이어가 항상 중력에 의한 Velocity를 갖고 있어서 문제로 보임
2. 플레이어 PQCS 관련한 문제로 보임.

클라이언트가 받는 Move패킷을 보니 움직이지 않아도 항상 플레이어 위치 패킷을 받고 있었다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Worker\_thread에서 IO\_TIMER\_PLAYER\_MOVE 메시지를 받아 처리하는 모습이다.

이 부분이 항상 실행되고 있었다는 의미는 Timer thread에서 PQCS로 IO\_TIMER\_PLAYER\_MOVE 메시지를 항상 보내고 있다는 뜻이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

PQCS로 IO\_TIMER\_PLAYER\_MOVE 메시지를 보내는 모습이다.

Length를 구하는 if문이 올바르지 않았으며,

Player의 Velocity의 크기가 매우 적거나 없으면 패킷을 보내지 않는다고 했지만 사실 Velocity의 크기는 중력으로 인해 일정 크기의 값을 항상 갖게 된다.

1. 물체 위에서의 플레이어 Jump

플레이어가 물체 위에 올라갈 수 있음에 따라 물체 위에서 Jump 할 수 있도록 하는 것이 자연스러울 것이다.

Terrain 바로 위에 있을 때 플레이어가 SPACE Key를 입력하면 Jump 하도록 구현해 놓은 부분을

변수 하나로 Jump를 할 수 있는지 없는지 판단하여 Jump를 수행하는 것으로 바꿔야 한다.



 

PlayerMovementComponent에 ‘Jump가 가능한가’를 판단하는 멤버 변수를 추가한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Jump 할 때 CanJump의 값으로 가능한지 판단하여 Jump를 수행한다.

Jump를 하고나서 다시 Jump하면 안되니 CanJump를 false로 바꾼다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

플레이어가 지형과 닿을 때를 판정하는 함수

지형에 닿았을 때 CanJump 값을 true로 바꿔 Jump 할 수 있게 한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

플레이어가 물체와 충돌되었을 때를 판정하는 함수

이 때 Obb의 윗면에 해당하는 부분과 플레이어가 충돌되면 지형과의 충돌처리와 유사하게 하여 중력에 의해 속도가 늘어나지 않게 하고 CanJump를 true로 바꿔준다.

위 작업으로 이제 물체 위에서 Jump 할 수 있다.