2D 게임에서의 중력 계산

남승현O, 방정원*
O*청강문화산업대학교 게임콘텐츠스쿨
e-mail: human1107@naver.comO, jwbang@ck.ac.kr*

Calculation of Gravity in a 2D Game

Seung-Hyeon NamO, Jung-Won Bang*
O*School of Game, Chungkang College of Cultural Industries

● 요 약 ●

게임에서 캐릭터가 점프 하는 중 플레이어가 스킬을 사용하면, 모든 물체가 정지 되는 기능을 구현해야 하는 상황에 놓이게 된다. Unity Engine에 내장 된 중력을 사용하면, 플레이어가 스킬을 사용 할 때 Rigid Body 속성을 사용하여 움직임을 제한할 수 있다. 그러나, 스킬 사용으로 인한 움직임정지를 해제 할 때 물체의 이전 속력이 사라져 움직임이 부자연스럽게 된다.
이를 해결하기 위해 수학 계산을 통해 시간 값에 따른 중력 값을 대입 하는 방법을 사용하면, 속력이 매우 커 타일을 통과해서 지나가는 현상이 나타난다. 본 논문에서는 다음 프레임 위치 계산을 통해 이러한 문제를 보정하는 방법과 수학 계산식을 통해 속력을 계산했을 때의 문제점 보완 방법 등에 대하여 연구하였다.

키워드: 강체(Rigid Body),중력 크기(Gravity Scale), 제약 조건(Constraints)

I. Introduction

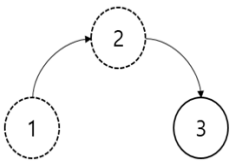

Fig. 1. 기대한 결과

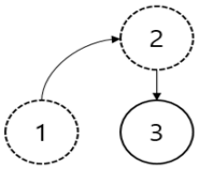

Fig. 2. 실제 결과

위의 그림은 플레이어가 오른 쪽으로 가면서 점프 했을 때의 상황을 가정한 그림이며 1번은 점프를 시작했을 때, 2번은 스킬을 사용했을 때, 3번은 스킬 사용이 끝나고 속력이 돌아왔을 때의 위치를 각각 그림으로 나타낸 것이다.

움직임을 제한 할 때 Rigid Body의 Gravity Scale과 Constraints으로 잠시 움직임을 멈춰 그림1)의 결과를 기대 했지만. 다시 움직임을 활성화 할 때 그림2)와 같이 이전의 속력이 사라져 부자연스러운 움직임이 발생하는 것을 볼 수 있었다.

이 논문에서는 수학식으로 시간에 따른 속력 값을 도출해 물체의 움직임을 프로젝트 구현을 통해 살펴보고, 이 문제를 보정하는 방법에 대하여 연구하였다.

II. Preliminaries

수학식으로 시간에 따른 속력 값을 도출해 물체의 움직임을 구현하였을 때, 프레임 이동의 문제점과 기본 중력의 문제점이 발생한다.

첫 번째로 프레임 이동의 예외 사항은 다음과 같다.

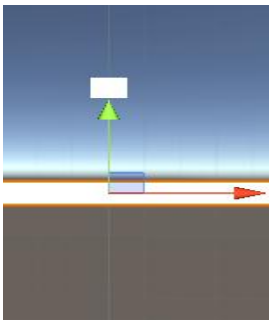

Fig. 3. 한 프레임 전

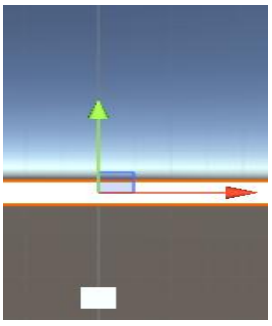

Fig. 4. 한 프레임 후

위 그림 속 Plane은 (0, 0) 물체는 (0, 5)에 위치해 있는 상태이다. (중력 -10p/s2, 시간 1/50s, 물체 속력 (0, -500)p/s) 한 프레임 후의 위치를 계산 해보면 dy = 500p/s * 1/50s 로 dy = 10p이며 위치는 (0, -5)가 된다. 프레임 이동은 값에 의한 이동 즉, 공간 이동이므로 Plane 과 충돌하지 않고 그림4)와 같이 (0, 5)에서 (0, -5) 로 이동함을 볼 수 있다. 이 예시로 알 수 있는 내용은 속력이 매우 클 경우 다른 물체를 통과한다는 것을 알 수 있다.

두 번째로 기본 중력의 문제점은 다음과 같다. 정해진 방향으로

날아가는 물체를 Rigid Body의 Constraints 기능을 사용해 물체가 잠시 멈추게 하면 해당 축의 속력이 0이 되어 그림 2와 같은 현상이 일어난다. 이 현상을 처리하려면 속성 값들을 다른 곳에 보관했다가 기능을 해제 할 때 다시 값을 가져와야 하는데 이렇게 할 경우 코드의 효율이 떨어지는 문제가 발생한다.

III. The Proposed Scheme

```
RaycastHit hit;
Physics.Raycast(transform.position, Vector3.down,out hit, 100.0f);

// 땅과 가깝거나 땅을 넘어서지 않고 Y축 속력이 점프 속도가 아닌 경우.
_isGround = (Mathf.Abs((hit.point - transform.position).y) <= 0.6f
    || (transform.position.y + (_rigidVelocity.y * Time.fixedDeltaTime))
    <= hit.point.y) && _rigidVelocity.y
    != _yJumpValue;

// isGround 가 False 인경우
if (!_isGround)
    // Y 축 속력 계산
    _rigidVelocity.y = _rigidVelocity.y +
        (_griavty * (Time.fixedDeltaTime * _time));

// isGround 가 True 인경우
else
{
    // 속력이 0 이 아닌 경우
    if (_rigidVelocity.y != 0.0f)
    {
        // 위치를 (BoxColider Y Size* 0.5 + hit.point.y) 만큼 보정해주며
        // Y축 속력을 0으로 만들어 줌.
        transform.position = new Vector3(transform.position.x,
            hit.point.y + 0.5f, 0);
        _rigidVelocity.y = 0.0f;
    }
}

// 최종 계산 속력을 대입
_rigid.velocity = _rigidVelocity * _time;
```

Fig. 5. 적용 코드

이 상황을 처리하는 방법으로 Engine 내장 기능인 Ray Cast를 사용 했다. 해당 방향에 물체가 맞았을 시 맞은 지점, 위치, 정보를 가져올 수 있게 되는데 이를 이용해서 바닥과의 거리가 가깝거나 다음 프레임 위치를 계산해서 다음 프레임 위치가 바닥을 지나쳤을 때 이를 보정해주는 처리를 하면 프레임 이동에 대한 예외 사항처리와 중력 계산 조건을 만들 수 있다.

Ray Cast를 통해 닿은 지점과 같거나 더 내려갔을 때 바닥에 닿았다고 체크를 하며 아닐 경우 계산식을 vf = vi + at를 사용해 Y 축의 속력을 계산하여 물체 속력에 대입하며 최종적으로 현재 Scene의 시간(0~1)을 곱해 최종 속도를 넣게 된다. 위와 같은 처리로 인해 물체가 땅에 닿아 있는 경우 중력에 의한 Y 축 속력 계산이 없는 것을 확인할 수 있었다. 속력이 -5000f/s 이지만 프레임 보정을 통해 바닥을 통과하여 지나가지 않음도 확인할 수 있었다.

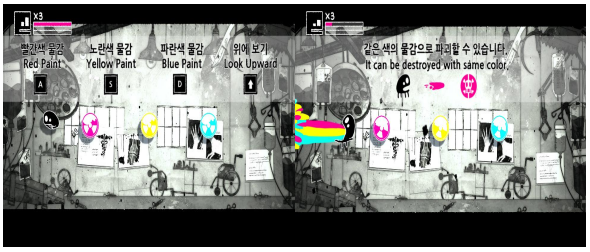

Fig. 6. Y 축 계산 멈추기 전/후

본 프로젝트에 이를 적용하여 스킬을 쓴 시점에 Y 축 계산이 되지 않는 것을 볼 수 있다.

IV. Conclusions

기본 물리엔진을 쓰지 않는 만큼 예기치 않는 상황들이 존재하지만 반대로 쓸 필요 없는 기능들을 덜어낼 수 있어 코드의 효율성을 높일수 있다. 그러나 예기치 못한 다양한 문제들을 직접 처리해 주어야 하는 데, 다양한 효과 구현을 위해 이에 대한 연구가 필요하다.

REFERENCES

[1] docs.unity3d.com/kr/530/ScriptReference/Physics.Raycast.html
[2] https://www.youtube.com/watch?v=9Y5W9cGCn10