객체지향 프로그래밍을 통한 물리엔진 개발

- 당구대의 동역학 -

2018 . 12 . 20

경희대학교 이과대학
물리학전공 오현식

지도교수 : 육순형

목차

I. 서론

컴퓨터들의 연산능력이 증가하면서, 과학 연구에서 시뮬레이션이 차지하는 비중은 점차 높아져왔다. 컴퓨터 시뮬레이션을 이용함으로서 이론과 실험결과의 비교분석이 용이해 졌고, 연구비를 절약할 수 있게 되었으며, 심지어 실행 불가능한 실험을 시뮬레이션을 통해 진행할 수도 있게 되었다. 특히 물리적 시뮬레이션은 약간의 변형을 거쳐 다른 시뮬레이터나 프로그램, 게임 내에 적용되기도 하는데, 이를 물리엔진이라고 한다.

물리엔진은 다른 프로그램 내에 삽입되어 단순히 현실감을 더해주는 수준부터 프로그램의 핵심기능을 차지하는 수준까지 다양한 역할을 수행한다. 물리엔진과 물리 시뮬레이션은 수행하는 역할이 같아 보이지만, 역할 수행의 목적과 방식에 차이가 있으며 이로 인해 그 구조 또한 달라진다.

첫째로 물리 시뮬레이션은 연산 시간에 제약이 없고, 전체 연산이 다 끝낸 후에 결과와 그 과정, 또는 경로를 한 번에 출력해준다. 그러나 물리엔진은 실시간으로 그래픽엔진, 게임엔진과 같은 프로그램의 다른 부분들과 정보를 주고받으며, 또한 대부분의 경우 이용자와 실시간으로 상호작용해야 한다. 따라서 하나의 긴 연산을 실행하는 시뮬레이션과 달리 물리엔진은 짧은 연산을 여러 번 실행하는 형태이며, 이에 따라 수초, 수분 뒤의 상황을 예측하는 것이 아닌 바로 다음 순간만을 예측하게 된다. 더불어 매 순간 그 결과를 출력해야하기 때문에 시뮬레이션 내의 1초를 연산하는데 현실의 1초 이상이 걸리면 안 된다.

둘째로 컴퓨팅 성능에 제약이 거의 없는 연구실에서의 시뮬레이션과 달리, 상업적 목적의 물리엔진은 사용자 PC 성능, 혹은 서버가 이용자마다 제공할 수 있는 최대 연산속도에 의해 그 연산속도가 제한된다. 따라서 물리엔진은 완벽한 결과를 추구하는 시뮬레이션과 달리 최대한 물리현상들을 단순화 시키면서도 현실과 유사한 결과를 보이는 최적화를 추구한다.

셋째로 물리엔진은 프로그램 상에서 항상 지배적인 위치를 차지하고 있는 것이 아니기 때문에, 필요에 따라 실행이 정지되거나, 불연속적이거나 비현실적인 운동을 구현할 필요성이 있다. 이를 위해선 프로그램의 다른 부분과 물리엔진이 적절하게 연동되도록 할 필요가 있다.

마지막으로 특정 실험에만 사용하기 위해 개발되는 경우가 많은 시뮬레이션과 달리, 물리엔진은 상업적인 목적으로 주로 사용되는 만큼, 다양한 고객이 다양한 상황에서 사용할 수 있도록 확장성과 유연성을 염두에 두고 개발된다.

초창기의 물리엔진은 단순히 움직임을 구현하는 수준에서 그쳤으나, 개인 PC 성능의 향상과 더불어 점차 자연스럽고 부드러운 움직임을 구현하게 되었고, 나아가 투사체에 적용되는 공기저항의 효과나, 광원효과까지 구현하게 되었다. 근래에는 영화산업의 3D 모델링이나 CG(Computer Graphics)분야, 게임산업, 가상 군사훈련, 산업공학의 개발과정 등에서 물리엔진이 차지하는 비중이 갈수록 커져가고 있으며, 이에 따라 아예 물리엔진만을 개발하여 상품으로 판매하는 업체들도 등장하고 있다.

본 연구에서는 물리현상에 대한 물리학적 분석이 어떻게 물리엔진으로 구현되는지 보이기 위해 일상에서 쉽게 접할 수 있는 가장 간단한 동역학적 시스템인 당구를 고전역학으로 분석하고 이를 물리엔진으로 구현하였다. 이를 위해여 객체지향 언어인 Python을 사용하였으며, 객체지향 언어가 가지는 확장의 용이성을 활용하여 물리엔진에 확장성을 부여할 수 있는 지 살펴보고, 물리엔진 개발과정에서 접할 수 있는 난관에는 무엇이 있는지 알아보고자 한다.

II. 물체에 작용하는 힘

1. 운동마찰력

마찰이 있는 평면 위에서 공이 운동할 때, 접점에서 작용하는 운동마찰력에 의해 물체의 속도와 각속도가 변화하게 된다. 각속도에 의해 생기는 접점의 무게중심에 대한 상대속도와, 물체의 선속도와의 합이 0이 된다면, 물체는 미끄러짐 없이 굴러가게 되고, 이 때 작용하는 마찰력은 0이다.

$$\vec{v} + \vec{\omega} \times (-R\hat{z}) = 0$$

만약 이 값이 0이 아니라면, 접점에서 테이블과 공 사이에 상대속도가 생기고, 그 반대방향으로 마찰력이 작용한다.

$$\vec{F} = -\mu mg \frac{\vec{v} + \vec{\omega} \times (-R\hat{z})}{|\vec{v} + \vec{\omega} \times (-R\hat{z})|}$$

이 힘은 동시에 돌림힘으로도 작용하며 이에 따른 속도와 각속도의 변화는 아래와 같다.

$$\vec{\tau} = (-R\hat{z}) \times \vec{F} = -\mu mg(-R\hat{z}) \times \frac{\vec{v} + \vec{\omega} \times (-R\hat{z})}{|\vec{v} + \vec{\omega} \times (-R\hat{z})|}$$

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \frac{1}{m}\vec{F} \qquad \frac{\mathrm{d}\vec{\omega}}{\mathrm{d}t} = \frac{1}{I}\vec{\tau} = \frac{5}{2mR^2}\vec{\tau}$$

2. 구름마찰력

바퀴나 공과 같은 물체가 미끄러짐 없이 굴러 간다면 앞서 본 바와 같이 정지마찰력이나 운동마찰력은 작용하지 않는다. 그럼에도 이들은 얼마안가 멈추는데, 이는 구름마찰력이 작용하기 때문이다. 구름마찰력은 굴러가는 물체에 작용하는 마찰력으로, 거친노면에 의한 효과, 또는 탄성을 가진 바닥이나 공이 그 중량에 의해 형태가 변형되었다가 복원되면서 손실되는 에너지 등을 뭉퉁그려 표현한 것이다. 따라서 구름마찰계수는 물체와 바닥의 재질, 상태에 따라 상수일 수도 있고 속도에 다양한 형태로 의존하는 값일 수도 있다. 여기서는 구름 마찰 계수를 상수 0.01로 잡았다.

구름마찰계수가 상수일 때 그 작용은 일반적인 마찰력과 거의 같다.

$$\vec{F} = -\mu mg\frac{\vec{v}}{|\vec{v}|} \qquad \vec{\tau} = (-R\hat{z}) \times \vec{F} = -\mu mg(-R\hat{z}) \times \frac{\vec{v}}{|\vec{v}|}$$

3. 그 외의 힘

일반적인 경우 당구공은 당구대 위에서만 움직인다. 따라서 z축 방향으로 작용하는 중력이나 수직항력과 같은 경우 대부분 서로 상쇄되므로 구현하지 않았다. 공기저항 역시 무의미할 정도로 작은 값이므로 생략한다.

III. 물체 사이의 충돌

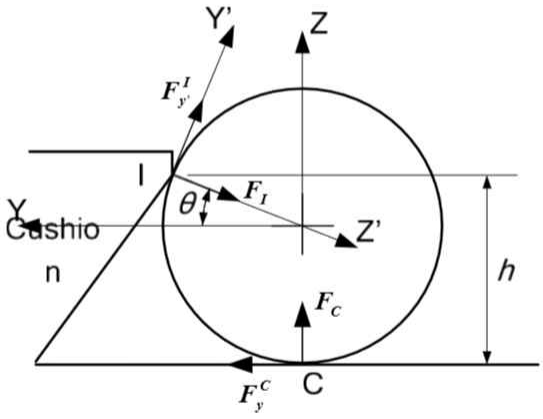
1. 벽과 공의 충돌

1) 굴러오는 공과 벽의 충돌

공과 고정된 벽이 충돌할 때의 운동량 변화는 탄성계수 e를 이용하면 쉽게 구할 수 있다. 변화하는 방향은 벽의 방향벡터와 같다.

$$\triangle\vec{v} = (1+e)(-\vec{v_0}\cdot\hat{n})\hat{n}$$

문제는 각속도이다. 우리는 경험적으로 미끄러짐 없이 굴러오는 공이 벽과 충돌하면 다시 미끄러짐 없이 굴러 나온다는 것을 알고 있다. 그러나 충돌과정에서 각속도에 변화를 줄 수 있는 힘은 마찰력뿐인데, 이는 각속도를 속도변화량에 발맞춰 바꾸기엔 충분치 않다. 그 비밀은 당구대의 구조에 있다.



<그림 1>[1)]

<그림 1>에서 보이듯, 당구대는 벽과 공의 충돌지점의 높이 h는 당구공 반지름 R에 대해 h = 7/5R의 값을 가지도록 설계되어 있다. 충돌지점 I에서 수평방향으로 충격력 F가 작용하여 물체의 속도와 위와 같이 변하였을 때, 운동량 변화량과 각속도 변화는 아래와 같다.

$$\sin\theta = \frac{2}{5}\ , \quad \triangle\vec{P} = m\triangle\vec{v} = m(1+e)(-\vec{v_0}\cdot\hat{n})\hat{n}$$

$$\triangle\vec{L} = \vec{r}\times\triangle\vec{P} = \vec{r}\times\hat{n}(\sin\theta\mid\triangle\vec{P}\mid) = \vec{r}\times\hat{n}(m\sin\theta\mid\triangle\vec{v}\mid) = I\triangle\vec{\omega}$$

$$I = \frac{2}{5}mR^2\ , \quad \frac{2}{5}mR\mid\triangle\vec{v}\mid = \frac{2}{5}mR^2\mid\triangle\vec{\omega}\mid$$

$$\therefore\mid\triangle\vec{v}\mid = R\mid\triangle\vec{\omega}\mid$$

---

1) Mathanvan S, Jackson M R, and Parkin R M, 2010. "A theoretical analysis of billiard ball dynamics under cushion impacts." p.5 Fig.3

이를 보면 미끄러짐 없이 굴러온 공은 마찬가지로 미끄러짐 없이 굴러나감을 알 수 있다. 위에 적용된 운동량 변화와 각속도 변화의 관계는 미끄러짐이 있는 경우에도 적용된다.

2) z축 방향 각 운동량에 의한 경로변경

공이 벽과 충돌하면서 접촉해 있는 짧은 시간 동안, 공과 벽의 접촉면에서도 운동마찰력이 작용한다. 만약 접점 I에서 벽면과 나란한 방향으로 상대속도가 존재한다면, 반대방향으로 운동마찰력이 작용하여 물체의 각속도와 속도를 바꾸게 된다. 이는 결과적으로 반사각이 입사각과 달라지는 결과를 낳게 된다. 바닥면과 공 사이에서 작용하는 운동마찰력과 달리, 여기서는 충격력이 수직항력을 대체하여 작용하게 된다. 우리는 충격력의 정확한 함수를 모르지만, 시간간격 △t 동안 충격력을 적분한 값이 운동량의 변화량임을 알기에, 이를 이용하여 마찰력에 의한 운동량과 각속도의 변화를 유도할 수 있다.

$$\int_{t0}^{t1} \vec{F}_{friction} dt = \int_{t0}^{t1} -\mu \frac{v_I}{|v_I|} \mid \vec{F}_{collision} \mid dt = -\mu m \frac{v_I}{|v_I|} \mid \triangle \vec{v}_{collision} \mid$$

$$\triangle \vec{v}_{friction} = -\mu \frac{v_I}{|v_I|} \mid (\triangle \vec{v})_{collision} \mid$$

$$\triangle \vec{\omega}_{friction} = -\mu \frac{m}{I} \mid (\triangle \vec{v})_{collision} \mid (-R\cos\theta \hat{n} \times \frac{v_I}{|v_I|})$$

2. 공 사이의 충돌

공 사이의 충돌은 완전탄성충돌에 가깝다.(e=0.95) 각각의 공의 질량이 완전히 같으므로, 식이 매우 단순화 시킬 수 있다. 두 공 사이에 있는 접촉면과 평행하게 x'축을, 수직하게 y'축을 설정하면 공1과 공2의 충돌 후 속도와 각속도는 아래와 같이 교환된다.

$$\hat{y'} = \frac{\overrightarrow{r1} - \overrightarrow{r2}}{|\overrightarrow{r1} - \overrightarrow{r2}|} \qquad (\vec{v}_1')_{y'} = e(\vec{v}_2)_{y'} \quad (\vec{v}_2')_{y'} = e(\vec{v}_1)_{y'} \quad (\vec{\omega}_1')_{x'} = e(\vec{\omega}_2)_{x'} \quad (\vec{\omega}_2')_{x'} = e(\vec{\omega}_1)_{x'}$$

속도의 x'성분과 각속도의 y'성분은 변하지 않고 유지된다.

Ⅳ. 물리엔진의 구현

1. 오일러 방법(Euler Method)

물리엔진은 매 순간 결과를 표시해 줘야하며, 프로그램의 다른 부분이나, 이용자와의 상호작용으로 인해 예측 불가능한 때에 물체의 속도나 가속도가 불연속적으로 변하곤 한다. 따라서 미분방적식을 푸는데에 오일러 방법을 쓰는 것이 가장 적절하다.

시간 t에 대한 물체의 위치벡터 r의 함수를 테일러 전개하면 다음과 같다.

$$\vec{r}(t+\triangle t) = \vec{r}(t) + \dot{\vec{r}}(t)\cdot\triangle t + \frac{1}{2!}\ddot{\vec{r}}(t)(\triangle t)^2 + \dots$$

3차항 이상을 0으로 두면 아래와 같이 정리할 수 있다.

$$\triangle\vec{r} = \vec{v}(t)\cdot\triangle t + \frac{1}{2}\vec{a}(t)(\triangle t)^2 \qquad \triangle\vec{v} = \vec{a}(t)(\triangle t)$$

위치벡터와 속도벡터, 가속도벡터, 각속도벡터는 numpy의 array클래스로 구현할 수 있으며, 앞서 구한 함수들 또한 numpy에서 제공하는 함수들로 구현할 수 있다. 물리엔진은 크게 세 부분으로 나누었다. 첫째는 충돌연산으로, 물체 간의 충돌을 감지하고, 충돌이 있을 시에 적절한 방법으로 속도와 각속도를 변화시킨다. 그 다음에는 가속도연산으로, 상황에 따라 물체에 작용할 힘을 선택하고, 그 힘들에 의해 공에 작용하는 가속도와 각가속도의 합을 구한다. 마지막은 운동연산으로, 앞서 구한 속도벡터와 가속도벡터를 이용하여 오일러 방법으로 다음 순간의 위치벡터를 구하고, 가속도에 의한 속도벡터의 변화 또한 계산한다.

2. 객체지향 프로그래밍

일일이 모든 물체에 해당하는 위치, 속도, 가속도, 각속도, 각가속도 벡터를 지정하고, 모든 힘과 충돌의 작용을 함수로 만들기보다는 당구공을 하나의 class로 만들어 각각의 공 객체 안에 변수들이 포함되도록 프로그래밍 하는 것이 효율적이다. 본 프로그램에서는 ball class와 wall class를 설정하고, 힘의 작용과 물체의 충돌을 각 객체나 변수 사이의 연산이 아닌, 클래스 사이의 연산으로 정의하였다. 선언한 물리 객체들의 리스트에 이 클래스간의 연산을 적용하는 방식으로 물리 엔진이 실행된다. 이 방식의 장점은 단 한 줄의 코드만으로 새로운 당구공, 새로운 벽을 추가할 수 있는 확장성에 있다.

3. 게임엔진과의 연결

간단하게 게임엔진을 구현할 수 있는 pygame 라이브러리를 이용하여 그 안에 물리엔진을 삽입하였다. 게임엔진은 크게 세 부분이 반복적으로 이어지는 구조를 가진다. 첫 번째는 이벤트 파트로 유저가 가하는 입력을 받아 상황에 맞게 처리하게 된다. 예를 들어 초기 속도와 각속도 값의 입력을 받고 그 수치를 물리엔진 내의 속도벡터와 각속도 벡터에 대입시키는 과정이 여기에 포함된다. 두 번째는 데이터 업데이트 파트로, 게임 내의 여러 데이터를 새로운 값으로 갱신시키는 연산을 진행한다. 물리엔진이 바로 여기에서 작동하며 각 물체의 위치와 속도를 갱신하게 된다. 마지막을 출력 파트로 갱신된 데이터에 따라 화면을 새로 랜더링하여 출력하거나 상황에 맞는 사운드를 출력하게 된다.

물리엔진은 사용자, 또는 다른 엔진과 데이터를 주고받으며 상호작용 하지 않으면 조잡한 시뮬레이션에 불과함을 잊지 말자. 보통 물리연산은 1초당 화면을 출력하는 횟수를 의미하는 fps(frame per second)에 맞춰서 시간간격 dt를 설정한다. 그러나 오일러법의 특성상 이 dt 값이 작을수록 더 정확한 연산이 가능하다. 따라서 게임엔진이 1회 반복되는 동안 물리엔진은 n회 실행되도록 계수를 통해 가속할 수 있도록 설정해 두었다. 갈수록 발전하는 그래픽 랜더

링 기술과 더불어 그래픽 랜더링이 요구하는 연산량도 증가하고 있다. 일반적으로 랜더링 연산은 그래픽카드에서 진행되고, 물리엔진을 포함한 나머지 연산은 cpu를 이용하기 때문에, 그래픽의 지연이 물리엔진의 정밀도에 영향을 주지 않게 하기 위해선 이러한 기능이 필요로 한다.

Ⅴ. 결과

1. 게임엔진과 물리엔진

물리엔진이 삽입된 게임엔진의 코드는 [부록]에 별첨했다.

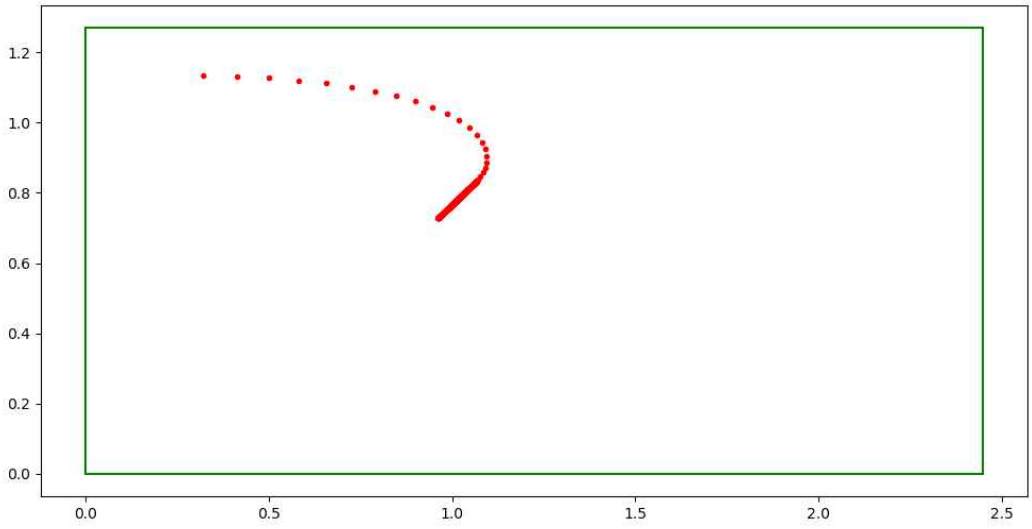
2. 물리엔진의 실행 결과

게임엔진 부분을 제거하고 물리엔진만 남긴 뒤 주어진 특정 상황에서 예상대로 운동하는지를 검증하였다.

1) 이동방향과 반대로 회전을 걸었을 때의 운동

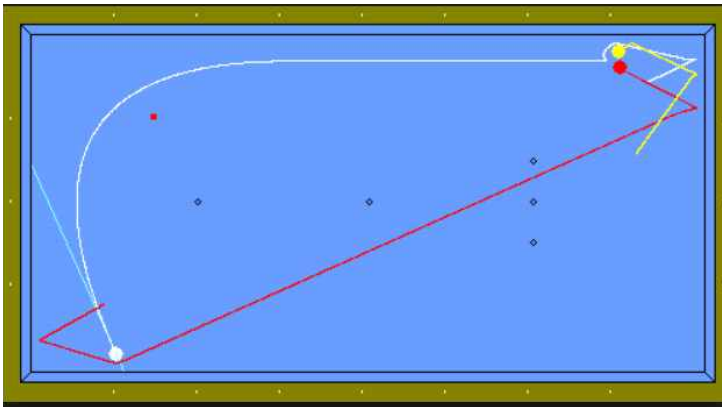
왼쪽 상단 위에 공을 두고 진행 방향과 반대로 회전을 걸었을 시, 운동마찰력에 의해 진행방향이 휘는 것이 구현되는 지 검증하였다. 초기 값은 아래와 같다.

위치 : r0 = [-1.0, 0.5, 0.0] (m)
속도 : v0 = [2.0, 0.0, 0.0] (m/s)
각속도 : w0 = [152.67, -305.34, 0.0] (rad/s)



<그림 2>

이는 아래와 같이 묘기당구에서 실제로 구현할 수 있는 궤적이다.

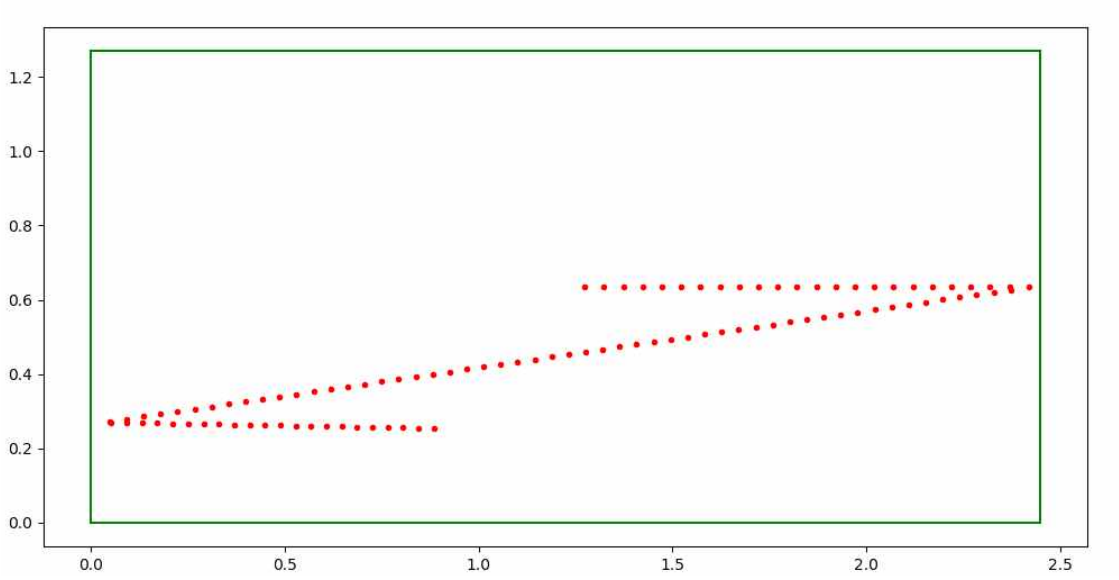


<그림 3>[2)]

2) 공과 벽의 충돌

z축 방향으로 회전을 가한 뒤, 공을 벽에 충돌시켜 튕겨져 나오는 경로가 변형되는 것을 보여준다.

위치 : r0 = [0.0, 0.0, 0.0] (m)
속도 : v0 = [1.0, 0.0, 0.0] (m/s)
각속도 : w0 = [0.0, 30.53, 91.60] (rad/s)



<그림 4>

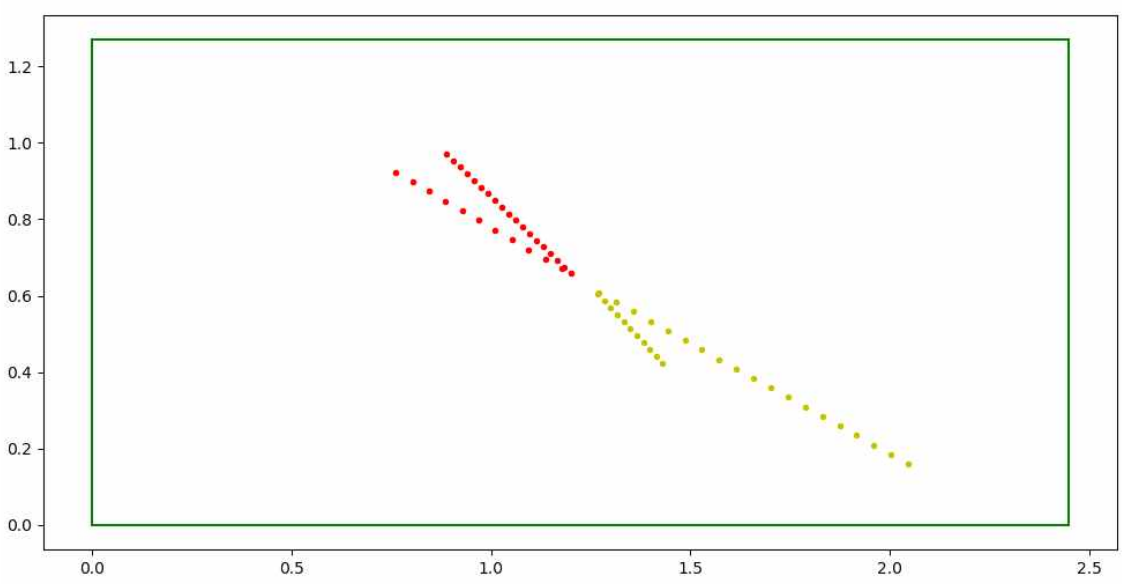
2) 2018년 12월 21일, http://www.writeopinions.com/artistic-billiards 참고

3) 공 사이의 충돌

<그림 5>에서 두 공이 거의 정면으로 충돌하는 경우, 서로 속도벡터가 교환되는 모습을 볼 수 있다.

빨간 공 : 1.0m/s, -45도 각도

노란 공 : 2.0m/s, 150도 각도

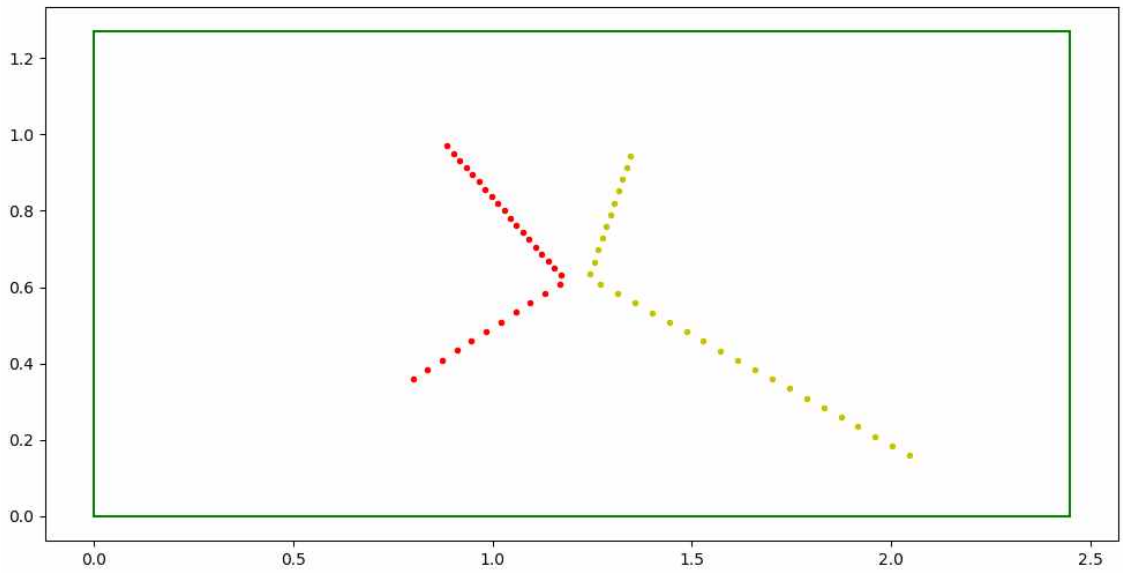


<그림 5>

여기서 빨간 공의 발사각을 -50도로 바꿔 비스듬하게 충돌시키면 <그림 6>과 같이 궤적이 변하는 것을 볼 수 있다.

빨간 공 : 1.0m/s, -50도 각도

노란 공 : 2.0m/s, 150도 각도



<그림 6>

※ 참고문헌

Gregory Jason, 『게임 엔진 아키텍처 : 게임 프로그래머가 꼭 알아야 할 게임 엔진 이론과 실무』, 박상희(역), 의왕:에이콘 출판, 2013

Stephen T Thornton, Jerry B. Marion, 『일반역학』, 강성태(역), 청범출판사, 2004

Grant R. Fowles, George L. Cassiday, 『해석역학』, 진병문(역), Cengage Leaning Korea Ltd. 2012

Yan-Bin Jia, Matthew T. Mason, Michael A. Erdmann, “Trajectory of a Billiard Ball and Recovery of Its Initial Velocities”

Mathanvan S, Jackson M R, and Parkin R M, “A theoretical analysis of billiard ball dynamics under cushion impacts.”, Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 2010, 224 (9), pp. 1863 - 1873.

※ 부록 : 게임엔진 코드

```
import numpy as np
import random

import pygame as pg
from pygame.locals import *
import sys

#물리엔진 관련
fps = 30
physical_accel = 6
pps = fps * physical_accel
table_len = 2.448
table_wid = 1.270
R = 0.03275  #당구공 반지름

ball_mass = 0.25 # 당구공 무게, kg



wall_wtrans = 0.1 # 벽과 공의 충돌 시 마찰 보정치
wall_e = 0.9 # 벽과 공의 탄성계수
wall_wtrans = 0.5 # 임시
ball_e = 0.98 # 공 사이의 탄성계수
rolling_u = 0.01 # 구름마찰계수
ball_table_u = 0.2 # 운동마찰계수
ball_ball_u = 0.05 # 공 사이의 운동마찰계수

ball_spin_dec = 10.0 #마찰에 의한 각속도 변화량 rad/s**2

g = 9.8 # 중력가속도 m/s**@

z_hat = np.array([0,0,1])
# 창크기 관련
edge = 50
size = 200.0 #table_len*size = 약 500
table_mid_x = edge + table_len*0.5*size
table_mid_y = edge + table_wid*0.5*size

#게임엔진 관련
pg.init()
screen = pg.display.set_mode((int(2.5*size + 2*edge), 600))
font = pg.font.SysFont('arial', 16)
clock = pg.time.Clock()
dodgerblue = pg.Color('dodgerblue2')
```

```
v_input = ''
wx_input = ''
wy_input = ''
wz_input =  ''
angle_input = ''
text_cursor = 0
dummy = 0
error = 0
errorcode = ''
working = 0

#색지정
color_red = (255,0,0)
color_green = (0,255,0)
color_blue = (0,0,255)
color_yellow = (255,255,0)
color_white = (255,255,255)

class ball():
    ball_list = []
    def __init__(self, name, mass, radius, color):
        self.name = name
        self.mass = mass
        self.radius = radius
        self.I = 2*(mass*(radius**2))/5
        self.classnum = 1
        ball.ball_list.append(self)
        # ball.ball_list += [self]
        self.r = np.zeros(3,float)
        self.v = np.zeros(3,float)
        self.a = np.zeros(3,float)
        self.w = np.zeros(3,float)
        self.alpha = np.zeros(3,float)
        self.color = color

    def random_location(self,n):
        if n==0:
            self.r = np.array([random.uniform(R,(table_len/2)-R),random.uniform(R,(table_wid/2)-R),R])
        elif n==1:
            self.r = np.array([-random.uniform(R,(table_len/2)-R),random.uniform(R,(table_wid/2)-R),R])
        elif n==2:
            self.r = np.array([-random.uniform(R,(table_len/2)-R),-random.uniform(R,(table_wid/2)-R),R])
        elif n==3:
            self.r = np.array([random.uniform(R,(table_len/2)-R),-random.uniform(R,(table_wid/2)-R),R])
        else:
            self.r = np.array([random.uniform(-table_len/2 + R, table_len/2 - R),random.uniform(-table_wid/2 + R,
                    table_wid/2 - R),R])
```

```
    def stop(self):
        self.v = np.zeros(3,float)
        self.a = np.zeros(3,float)
        self.w = np.zeros(3,float)
        self.alpha = np.zeros(3,float)

    def shoot(self, shoot_v, shoot_angle, shoot_wx, shoot_wy, shoot_wz):
        # wx와 wy는 vx와 vy를 기준으로 한다. [wx] x [wy] = - ( vx + vy)
        # 또한 그 단위는 각속도가 아닌, 구 표면의 속도를 m/s단위로 한다.
        if shoot_v == 0.0:
            shoot_v = 0.0001
        self.v = np.array([shoot_v*np.cos(shoot_angle),shoot_v*np.sin(shoot_angle),0.])
        self.w = np.array([0.,0.,shoot_wz/R])
        self.w += (shoot_wx/R)*normal(np.array([-shoot_v*np.sin(shoot_angle),shoot_v*np.cos(shoot_angle),0.0]))
        self.w += (shoot_wy/R)*normal(self.v)


class wall():
    wall_list = []
    def __init__(self, name, a, b, c, d):
        self.name = name
        self.a = a
        self.b = b
        self.c = c
        self.d = d
        self.classnum = 2
        wall.wall_list += [self]
        self.N = np.array([self.a , self.b , self.c])/np.linalg.norm(np.array([self.a , self.b , self.c]))

    # 무한벽의 방정식은 wall.fuction() = 0
    def function(self, x, y, z):
        return ((self.a)*x + (self.b)*y + (self.c)*z + self.d)

    def distance(self, ra):
        dist = np.abs(self.function(ra[0],ra[1],ra[2])) / np.sqrt(self.a**2 + self.b**2 + self.c**2)
        return dist

# 함수정의

def normal(v):
    norm=np.linalg.norm(v)
    if norm==0:
        norm=np.finfo(v.dtype).eps
    return v/norm
```

```
def crash_detection(A,B):
    result=[0,0]
    if (A.classnum ==2) and (B.classnum==2):
        result = [2,2]
    elif (A.classnum ==1) and (B.classnum==1):
        if np.abs(np.linalg.norm(A.r - B.r)) <= (A.radius + B.radius):
            result = [1,1]
    elif (A.classnum ==1) and (B.classnum ==2):
        if B.distance(A.r) < A.radius:
            result = [1,2]
    elif (A.classnum == 2) and (B.classnum == 1):
        if A.distance(B.r) < B.radius:
            result = [2,1]
    else:
        result = [0,0]

    return result

def moving():
    for a in range(0,len(ball.ball_list)):
        A = ball.ball_list[a]
        A.v[2] = 0.0
        A.r += (A.v * (1.0/pps)) + (0.5)*A.a*((1/pps)**2)
        A.v += A.a * (1/pps)
        A.v[2] = 0.0
        A.w += A.alpha * (1/pps)

def ball_locating():
    n = 0
    N = list(range(0,len(ball.ball_list)))
    random.shuffle(N)
    for obj in ball.ball_list:
        obj.random_location(N[n]%4)
        n+=1

def ball_draw():
    for a in range(0,len(ball.ball_list)):
        A = ball.ball_list[a]
        pg.draw.circle(screen,A.color,(int(table_mid_x + A.r[0]*size),int(table_mid_y
        - A.r[1]*size)),int(A.radius*size))
```

```
def crashing():
    for a in range(0,len(physics_list)):
        for b in range(a+1,len(physics_list)):
            A = physics_list[a]
            B = physics_list[b]
            if crash_detection(A,B) == [1,2]:
                bf_w = np.copy(A.w)
                bf_v = np.copy(A.v)
                A.w += ((1+wall_e)/R)*((-1)*np.inner(bf_v , B.N))*np.cross(z_hat,B.N)
                A.v += (1+wall_e)*((-1)*np.inner(bf_v , B.N))*B.N
                # 마찰의 효과
                dv = (-1)*(np.sqrt(21.0/25.0)*R)*np.cross((np.inner(bf_w,z_hat)*z_hat),B.N) + (bf_v -
                      np.inner(bf_v,B.N)*B.N)
                if abs(np.linalg.norm(dv)) > 0.0:
                    dp = A.mass*(1+wall_e)*((-1)*np.inner(bf_v , B.N))*ball_table_u*wall_wtrans
                    if dp >= abs(np.linalg.norm(dv)):
                        dp = 0.5*abs(np.linalg.norm(dv))
                    dp *= (-1)*normal(dv)
                    A.v += (dp/A.mass)
                    A.w += np.cross(((-1)*R*np.sqrt(21.0/25.0)*B.N), dp)/A.I
                A.v[2] = 0.0
                A.r += 0.5*A.v*(1/pps)
            elif crash_detection(A,B) == [2,1]:
                bf_w = np.copy(B.w)
                bf_v = np.copy(B.v)
                B.w += ((1+wall_e)/R)*((-1)*np.inner(bf_v , A.N))*np.cross(z_hat,A.N)
                B.v += (1+wall_e)*((-1)*np.inner(bf_v , A.N))*A.N
                # 마찰의 효과
                dv = (-1)*(np.sqrt(21.0/25.0)*R)*np.cross((np.inner(bf_w,z_hat)*z_hat),A.N)
                       + (bf_v - np.inner(bf_v,A.N)*A.N)
                if abs(np.linalg.norm(dv)) > 0.0:
                    dp = B.mass*(1+wall_e)*((-1)*np.inner(bf_v , A.N))*ball_table_u*wall_wtrans
                    if dp >= abs(np.linalg.norm(dv)):
                        dp = 0.5*abs(np.linalg.norm(dv))
                    dp *= (-1)*normal(dv)
                    B.v += (dp/B.mass)
                    B.w += np.cross(((-1)*R*np.sqrt(21.0/25.0)*A.N), dp)/B.I
                B.v[2] = 0.0
                B.r += 0.5*B.v*(1/pps)
            elif crash_detection(A,B) == [1,1]:
                r_hat = normal(B.r - A.r)
                var = np.inner(A.v, r_hat)*r_hat
                vap = A.v - var
                wap = np.inner(A.w, np.cross(z_hat,r_hat))*np.cross(z_hat,r_hat)
                vbr = np.inner(B.v, r_hat)*r_hat
                vbp = B.v - vbr
                wbp = np.inner(B.w, np.cross(z_hat,r_hat))*np.cross(z_hat,r_hat)
                A.v = vap + ball_e*vbr
                A.w += (-1)*wap + ball_e*wbp
                B.v = vbp + ball_e*var
                B.w += (-1)*wbp + ball_e*wap
                A.r += 0.25*A.v*(1/pps)
                B.r += 0.25*B.v*(1/pps)
```

```
def force():
    # 가속도를 전부 초기화 한 뒤, 다시 부여한다.
    for a in range(0,len(ball.ball_list)):
        A = ball.ball_list[a]
        A.a = np.zeros(3,float)
        A.alpha = np.zeros(3,float)
    for a in range(0,len(ball.ball_list)):
        A = ball.ball_list[a]
        # 본래는 여기서 조건문으로 어떤 힘을 주어야 할지 정해야한다.
        # 그러나 본 시뮬에서는 모든 공이 테이블 위에서만 움직이기에 항상 같은 종류의 힘만 받으므로 생략
        # 운동마찰력
        point_v = A.v + (-R)*np.cross(A.w, z_hat)
        if abs(np.linalg.norm(point_v))>0.0:
            if ball_table_u*g*(1/pps) >= abs(np.linalg.norm(point_v)):
                movingfriction = (-0.5)*point_v*pps
            else:
                movingfriction = A.mass*ball_table_u*g*(-1)*normal(point_v)
            A.a += movingfriction/A.mass
            A.alpha += (1/A.I)*np.cross(((-R)*z_hat), movingfriction)
        # 구름마찰력
        if abs(np.linalg.norm(A.v)) != 0.0 :
            if rolling_u*g*(1/pps)*(7.0/abs(np.linalg.norm(A.v))) <= np.linalg.norm(A.v):
                rollingfriction = (-1)*rolling_u * A.mass * g *normal(A.v)*(7.0/abs(np.linalg.norm(A.v)))
                A.a += rollingfriction/A.mass
                A.alpha += np.cross(((-R)*z_hat), rollingfriction)/A.I
            else:
                rollingfriction = (-1)*A.mass*A.v*pps
                A.a += rollingfriction/A.mass
                A.alpha += np.cross(((-R)*z_hat), rollingfriction)/A.I

        # 마찰에 의한 z축 회전 감소
        if np.inner(A.w, z_hat) >= ball_spin_dec*(1/pps):
            A.alpha += (-1)*ball_spin_dec * normal(np.inner(A.w,z_hat)*z_hat)
        else:
            A.alpha += (-1)*np.inner(A.w,z_hat)*z_hat * pps
```

```
#객체 생성

west = wall("west",1.,0.,0.,(0.5*table_len))
east = wall("east",-1.,0.,0.,(0.5*table_len))
north = wall("north",0.,-1.,0.,(0.5*table_wid))
south = wall("south",0.,1.,0.,(0.5*table_wid))

yellow1 = ball("yellow1",ball_mass,R,color_yellow)
yellow2 = ball("yellow2",ball_mass,R,color_yellow)
red1 = ball("red1",ball_mass,R,color_red)
white1 = ball("white1",ball_mass,R,color_white)


#객체 리스트
physics_list = ball.ball_list + wall.wall_list

#ball_locating()
white1.random_location(1)

#실행
while True:
    #이벤트(입력)을 받는다

    for event in pg.event.get():
        if event.type == pg.QUIT:
            #pg.quit()
            #sys.exit()
            break
        # 값을 입력중이지 않을 때, 입력을 선택하거나, 이미 입력된 값을 집어넣는다.
        if event.type == pg.KEYDOWN:
            if text_cursor == 0:
                if event.key == pg.K_LEFT:
                    text_cursor = 1

                elif event.key == pg.K_RIGHT:

                    text_cursor = 1
                    v_input = ''
                    wx_input = ''
                    wz_input =  ''
                    angle_input = ''

                elif event.key == pg.K_DOWN:
                    #멈추고, 재위치
                    for obj in ball.ball_list:
                        obj.stop()
                    ball_locating()
```

```
            elif event.key == pg.K_SPACE:
                #멈추기만
                for obj in ball.ball_list:
                    obj.stop()

            elif event.key == pg.K_RETURN:
                # 입력중이지 않은 상태에서 엔터를 누르면 입력된 값이 대입된다.
                # 하지만 입력값이 float으로 변환불가능하면 에러메세지 출력. 다시 입력해야함.
                if working==0:
                    try:
                        white1.shoot(float(v_input), 2*np.pi*float(angle_input)/360, float(wx_input),
                          float(wy_input), float(wz_input))
                        error = 0
                        working =1
                    except ValueError:
                        error = 1
                        print("wrong input")

        #속도입력
        if text_cursor == 1:
            if event.key == pg.K_RETURN:
                text_cursor = 2
            elif event.key == pg.K_BACKSPACE:
                v_input = v_input[:-1]
            else:
                v_input += event.unicode
        #옆방향각속도입력
        elif text_cursor == 2:
            if event.key == pg.K_RETURN:
                text_cursor = 3
            elif event.key == pg.K_BACKSPACE:
                wx_input = wx_input[:-1]
            else:
                wx_input += event.unicode

        #옆방향각속도입력
        elif text_cursor == 3:
            if event.key == pg.K_RETURN:
                text_cursor = 4
            elif event.key == pg.K_BACKSPACE:
                wy_input = wy_input[:-1]
            else:
                wy_input += event.unicode
```

```
            #수직방향각속도입력
            elif text_cursor == 4:
                if event.key == pg.K_RETURN:
                    text_cursor = 5
                elif event.key == pg.K_BACKSPACE:
                    wz_input = wz_input[:-1]
                else:
                    wz_input += event.unicode
            #발사방향 입력
            elif text_cursor == 5:
                if event.key == pg.K_RETURN:
                    text_cursor = 0
                    working = 0
                elif event.key == pg.K_BACKSPACE:
                    angle_input = angle_input[:-1]
                else:
                    angle_input += event.unicode

    # 데이터 연산(물리엔진) 실시
    for tt in range(0,physical_accel):
        crashing()
        force()
        moving()

    #에러문구
    if error == 0:
        errorcode = ''
    elif error == 1:
        errorcode = 'wrong input'


    # 화면갱신
    # 화면채우기
    screen.fill(color_white)

    # 입력란
    v_input_surface = font.render('    V     (m/s) : ' + v_input, True, dodgerblue)
    wx_input_surface = font.render('Wx * R (m/s) : ' + wx_input, True, dodgerblue)
    wy_input_surface = font.render('Wy * R (m/s) : ' + wy_input, True, dodgerblue)
    wz_input_surface = font.render('Wz * R  (m/s) : ' + wz_input, True, dodgerblue)
    angle_input_surface = font.render(' angle  (`)   : ' + angle_input, True, dodgerblue)
    errorcode_surface = font.render(errorcode,True,dodgerblue)
    screen.blit(v_input_surface, (50, 400))
    screen.blit(wx_input_surface, (50, 430))
    screen.blit(wy_input_surface, (50, 460))
    screen.blit(wz_input_surface, (50, 490))
    screen.blit(angle_input_surface, (50, 520))
    screen.blit(errorcode_surface, (50, 550))
```

```
    # 상태란(속도절댓값, 각속도)
    v_track_surface = font.render('    V      (m/s) : ' + str(abs(np.linalg.norm(white1.v)))), True,
                          dodgerblue)
    w_track_surface = font.render('    W*R   (m/s) : ' + str(abs(R*np.linalg.norm(white1.w)))), True,
                          dodgerblue)
    wz_track_surface = font.render('    Wz*R   (m/s) : ' + str(abs(R*np.inner(white1.w,z_hat)))), True,
                           dodgerblue)

    screen.blit(v_track_surface, (250, 400))
    screen.blit(w_track_surface, (250, 430))
    screen.blit(wz_track_surface, (250, 460))

    #테이블
    pg.draw.rect(screen,color_green,(edge,edge,int(table_len*size),int(table_wid*size)))
    #pg.draw.circle(screen,color_white,(int(table_mid_x + white1.r[0]*size),int(table_mid_y
                    - white1.r[1]*size)),int(R*size))
    ball_draw()

    #화면출력
    pg.display.flip()
    clock.tick(fps)
pg.quit()
sys.exit()
```