프로젝트 주제	레이놀즈 수 기반 항력과 유체역학적 힘(양력, 부력 등) 모델을 적용한 파이썬 강체 운동 시뮬레이션 엔진

프로젝트 진행 방법 (사용한 도구, 프로그램, 과정 등)

언어 / 운영체제	Python 3.8.10 / Windows 11
라이브러리	물리엔진: math, numpy, abc, dataclasses, typing 렌더링: pygame, numpy

클래스 객체

World			
위치	core/world.py		
상속	object		
기능	물리 엔진 연산 세계 구성 및 step 실행 (시뮬레이션 전체 컨테이너)		
메서드	이름	인자: 타입	기능
	__init__	self, background: Background, C_rot: float	배경(공기나 물 등으로 이루어진 세계) 및 회전 감쇠 계수 정의
	add_body	self, body: Body	강체 추가
	step	self, dt: float	외력 적용 -> 적분 계산 -> 충돌 처리(검사 -> 보정)

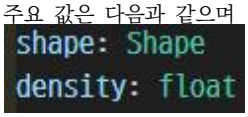
Shape			
위치	core/share.py		
상속	abc.ABC		
기능	물체의 기하학적 형태 추상 클래스		
메서드	이름	인자: 타입	기능
	volume	self	부피[m^3]
	ref_area	self	항력계수/항력/양력 계산에 쓰이는 기준 면적(A0)[m^2]
	char_length	self	레이놀즈 수 계산에 쓰이는 물체의 길이 [m]
	spin_radius	self	스핀파라미터 계산에 쓰이는 물체의 길이 [m]
	Izz_from_mass	self, mass: float	z축 관성 모멘트 [kg*m^2]
	proj_area	self, yaw_rad: float, velocity: numpyndarray	투영 단면적 [m^2]

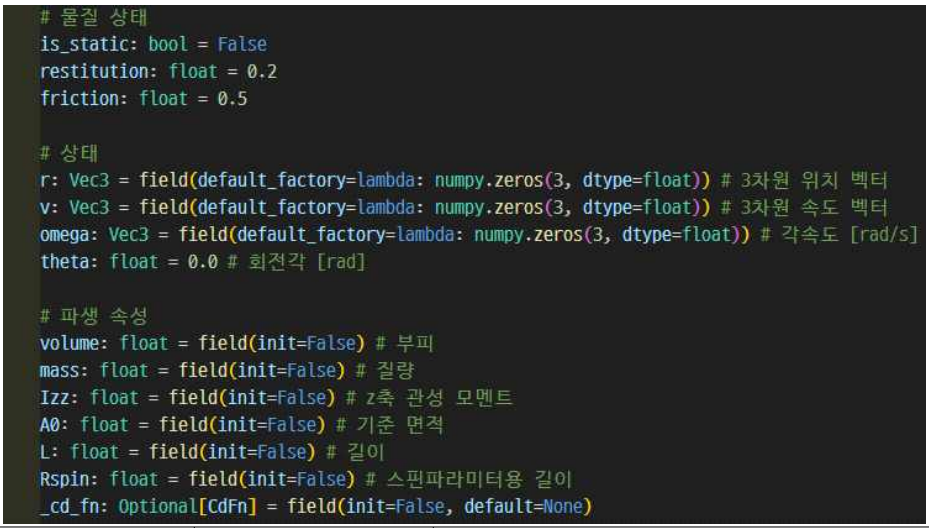
Sphere			
위치	core/shape.py		
상속	Shape		
데코레이터	@dataclass(frozen=True)		
기능	구 형태 데이터 클래스 정의		
메소드	이름	인자: 타입	기능
	매소드 오버라이딩 하였으므로 부모 클래스와 동일함		

Box			
위치	core/shape.py		
상속	Shape		
데코레이터	@dataclass(frozen=True)		
기능	육면체 형태 데이터 클래스 정의		
메서드	이름	인자: 타입	기능
	메서드 오버라이딩 하였으므로 부모 클래스와 동일함		

Body			
위치	core/body.py		
상속	abc.ABC		
기능	물체의 기하학적 형태 추상 클래스		
메서드	이름	인자: 타입	기능
	state	self	상태 벡터 반환"
	set_state	self	상태 벡터 설정
	drag_coefficient	self, y: numpy.ndarray	항력 계수 반환
	proj_area	self	속도 벡터 방향의 투영 단면적 반환

<table>
<tr><td></td><td>
<table>
<tr><th colspan="4">RigidBody</th></tr>
<tr><td>위치</td><td colspan="3">core/body.py</td></tr>
<tr><td>상속</td><td colspan="3">Body</td></tr>
<tr><td>기능</td><td colspan="3">3D 전용 강체
- 3차원 벡터를 기본적으로 포함함
- 회전각과 각속도는 z축에서만 정의됨, 다른 축은 0.0
!! 따라서 z축 위치와 속도 및 각속도(x축, y축)는 0.0으로 정의해야함(다른 값 넣으면 _enforce_2p5d_constraints를 통해 강제로 조정됨)
- 부피, 질량 등은 자동으로 계산됨. 만약 직접 질량을 조정하고 싶으면 CustomRigidBody를 사용하면 됨 상태벡터는 다음과 같이 구성됨 [x,y,z, vx,vy,vz, ωx,ωy,ωz, theta] 각
각 위치벡터, 병진속도벡터, 각속도벡터, 회전각임.

주요 값은 다음과 같으며
 

다음의 값을 가지고 있음.
  </td></tr>
<tr><td rowspan="5">메서드</td><td>이름</td><td>인자: 타입</td><td>기능</td></tr>
<tr><td>__post_init__</td><td>self</td><td>값 정의</td></tr>
<tr><td>_enforce_2p5d_constraints</td><td>self</td><td>z축 이동 및 xy축 회전 제한</td></tr>
<tr><td>_bind_drag_coefficient</td><td>self,</td><td>항력 계수 반환 함수 설정</td></tr>
<tr><td colspan="3">이 외는 메서드 오버라이딩 하였으므로 부모 클래스와 동일함</td></tr>
</table>
<table>
<tr><th colspan="4">CustomRigidBody</th></tr>
<tr><td>위치</td><td colspan="3">core/body.py</td></tr>
<tr><td>상속</td><td colspan="3">RigidBody</td></tr>
<tr><td>기능</td><td colspan="3">질량을 밑 관성 모멘트를 임의로 설정할 수 있는 강체 클래스</td></tr>
<tr><td rowspan="2">메서드</td><td>이름</td><td>인자: 타입</td><td>기능</td></tr>
<tr><td colspan="3">이 외는 메서드 오버라이딩 하였으므로 부모 클래스와 동일함</td></tr>
</table>
</td></tr>
</table>

Background			
위치	core/background.py		
상속	object		
기능	배경 유체 정의 클래스		
메서드	이름	인자: 타입	기능
	__init__	self, Density, Absolute_Viscosity, Kinematic_Viscosity	밀도, 점성 계수(절대 점도), 동점성 계수를 가지고 있는 데이터 클래스 역할을 함.

Event			
위치	core/event.py		
상속	object		
기능	충돌 정보 클래스		
메서드	이름	인자: 타입	기능
	__init__	self, body_a: Body, body_b: Body	충돌한 두 물체의 정보와 접촉점(self.contacts) 리스트, 충돌 법선(self.normal), 침투 깊이(self.penetration) 정보를 저장함.

CD			
위치	core/drag_coefficient.py		
상속	object		
기능	레이놀즈 수에 따른 항력계수를 반환하는 함수 모음(@staticmethod 매서드 모음임)		
메서드	@데코레이터 함수 이름	인자: 타입	기능
	@staticmethod sphere	Re: float	레이놀즈 수에 따른 구의 항력 계수
	@staticmethod cube	Re: float	레이놀즈 수에 따른 정육면체의 항력 계수
	@staticmethod box	Re: float	레이놀즈 수에 따른 직육면체의 항력 계수

Force			
위치	core/force.py		
상속	object		
기능	병진 운동에 관한 힘 계산 함수 모음(@staticmethod 매서드 모음임)		
메서드	@데코레이터 함수 이름	인자: 타입	기능
	@staticmethod gravity	body: Body	중력 계산
	@staticmethod buoyancy	density_fluid: float, body: Body	부력 계산(완전 잠긴 상태 기준임)
	@staticmethod drag	density_fluid: float, body: Body, Re: float	항력 계산
	@staticmethod lift	density_fluid: float, body: Body,	표준형 양력(마그누스 힘) F = 0.5*밀도*단면적*CL*v^2 * n_perp 3D(평면 운동 + z축 회전) 가정 omega는 스칼라(omega_z)만 사용.

Torque			
위치	core/force.py		
상속	object		
기능	회전 운동에 관한 토크(돌림 힘) 계산 함수 모음(@staticmethod 매서드 모음임)		
메서드	@데코레이터 함수 이름	인자: 타입	기능
	@staticmethod Damping	C_rot: float, body: Body	회전 감쇠 토크 계산 (고속에서 비선형 기준임) = 회전 감쇠율 x |각속도| x 각속도

Density		
위치	config.py	
상속	object	
기능	각종 물체의 밀도 데이터 모음(섭씨 20도 및 표준압(1 atm = 101325 Pa) 기준 / 습도 무시)	
하위 클래스	클래스 이름	기능

	Gas	# 대기 구성 성분
		Air = 1.204 # 대기
		Nitrogen = 1.165 # 질소
		Oxygen = 1.331 # 산소
		Argon = 1.661 # 아르곤
		Carbon_Dioxide = 1.842 # 이산화탄소
		
		# 가벼운 기체
		Hydrogen = 0.084 # 수소
		Helium = 0.166 # 헬륨
		Neon = 0.838 # 네온
		
		# 연료 및 기타 기체
		Methane = 0.668 # 천연가스(메테인)
		Propane = 1.882 # 석유가스(프로판)
		Butane = 2.489 # 뷰테인
		Ammonia = 0.717 # 암모니아
		Chlorine = 2.994 # 염소
		Radon = 9.23 # 라돈
		Steam_100C = 0.597 # 수증기 (섭씨 100도 기준)

	Liquid	class Water:
		Pure = 998.207
		Sea = 1025 # 바닷물 (1020~1030)
		Distilled = 997 # 증류수
		Heavy = 1105 # 중수 (D2O)
		
		class Fuel:
		Gasoline = 740 # 휘발유 (720~775)
		Diesel = 830 # 경유 (820~860)
		Kerosene = 810 # 등유
		Jet_Fuel = 800 # 항공유 (Jet A-1)
		LNG_Liquid = 447 # 액화천연가스 (액화메테인)
		LPG_Liquid = 510 # 액화석유가스 (액화프로판)
		
		class Alcohol:
		Ethanol = 789 # 에탄올
		Methanol = 792 # 메탄올
		Isopropyl = 786 # 이소프로필 알코올
		
		class Oil:
		Olive = 915 # 올리브유
		Vegetable = 920 # 식용유 (대두유)
		Engine_10W30 = 875 # 엔진오일 (870~890)
		Crude_Light = 850 # 경질 원유
		Crude_Heavy = 970 # 중질 원유
		
		class Household_Chemical: # 가정용 화학물질
		Glycerin = 1261 # 글리세린
		Honey = 1420 # 꿀 (1400~1450)
		Milk = 1030 # 우유
		Vinegar = 1010 # 식초
		Mercury = 13546 # 수은
		Blood = 1060 # 혈액 (인간)

	Solid	class Wood: # 섭씨 21도 65%습도의 12% 함수율 기준 https://blog.naver.com/robingoody/220939148564 출처
		Zelkova_Serrata = 720 # 느티나무 (13% 함수율)
		Pinus_Densiflora = 460 # 소나무
		Quercus_Acutissima = 750 # 한국 참나무(참나무속 상수리나무)
		Quercus_Rubra = (red_oak := 650) # 루브라참나무(레드 오크)
		Quercus_Alba = (White_Oak := 640) # 흰참나무(화이트 오크)
		Cherry = 540 # 벚나무
		American_Walnut = 580 # 호두나무
		Birch = 610 # 자작나무
		Sweet_Chestnut = 540 # 밤나무
		Beech = 710 # 너도밤나무
		Hard_Maple = 640 # 단풍나무
		Persimmon = 780 # 감나무
		Cedar = 320 # 동양 삼나무
		Guibourtia_Tessmannii = 910 # 구이부르티아 테스마니(부빙가)
		Horse_Chestnut = 490 # 마로니에(가시칠엽수)
		Dalbergia_Latifolia = (Indian_Rosewood := 830) # 북인도자단나무(북인도황단)
		
		class Material: # 금속
		class Iron: # 철류
		Pure = 7870 # 순철
		Wrought = 7800 # 연철 (7800~7870)
		Steel = 7850 # 강철(탄소강 기준)
		Cast = 7200 # 주철(무쇠 7100~7300)
		Pig = 7100 # 선철(무쇠 7000~7200)
		class Alloy: # 합금
		Stainless_Steel = 7930 # 스테인리스강
		Brass = 8500 # 황동 (구리+아연, 8400~8700)
		Bronze = 8800 # 청동 (구리+주석, 8700~8900)
		Duralumin = 2800 # 두랄루민 (강력 알루미늄 합금)
		Inconel = 8250 # 인코넬 (니켈 합금)
		class Common: # 일반 금속
		Aluminum = 2700 # 알루미늄
		Copper = 8960 # 구리
		Lead = 11340 # 납
		Nickel = 8900 # 니켈
		Zinc = 7133 # 아연
		Tin = 7280 # 주석
		Titanium = 4506 # 티타늄 (강철보다 가볍고 강함)
		Magnesium = 1738 # 마그네슘 (실용 금속 중 가장 가벼움)
		class Precious_Metal: # 귀금속
		Gold = 19300 # 금
		Silver = 10490 # 은
		Platinum = 21450 # 백금
		Palladium = 12020 # 팔라듐
		class Heavy_Metal: # 중금속
		Tungsten = 19250 # 텅스텐 (매우 무거움)
		Mercury = 13546 # 수은 (상온 액체 상태)
		Uranium = 19050 # 우라늄
		Osmium = 22570 # 오스뮴 (지구상에서 가장 밀도가 높은 원소)
		
		class Rock: # 돌
		class Igneous: # 화성암
		Granite = 2700 # 화강암 (2600~2800)
		Andesite = 2695 # 안산암 (2250~3140)
		Basalt = 2750 # 현문암
		class Sedimentary: # 퇴적암
		Mudstone = 2600 # 이암
		Shale = 2600 # 셰일(혈암)
		Sandstone = 2500 # 사암 (2200~2800)
		Conglomerate = 2650 # 역암 (2500~2800)
		Limestone = 2100 # 석회암 (1500~2700, 석회암)
		Tuff = 2450 # 응회암 (1600~3300)
		Chert = 2800 # 각암
		Halite = 2168 # 암염
		Dolomite = 2950 # 백운암 (2900 ~ 3000,)
		class Metamorphic: # 변성암
		Marble = 2720 # 대리석
		Quartzite = 2665 # 규암
		Slate = 2700 # 점판암
		Schist = 2850 # 편암
		Gneiss = 2750 # 편마암
		Quartz = 2650 # 석영(이산화규소)

Absolute_Viscosity		
위치	config.py	
상속	object	
기능	점성 계수(절대 점도) μ (섭씨 20도 기준) [Pa・s]	
내부 데이터	이름	기능
	PureWater	물 점성 계수
	Air	공기 점성 계수

Kinematic_Viscosity		
위치	config.py	
상속	object	
기능	동점성 계수 $\nu = \mu / \rho$ [m^2/s]	
내부 데이터	이름	기능
	PureWater	물 동점성 계수 1.003e-06
	Air	공기 동점성 계수 1.156e-5

World 구조 (config 등 상수 정의 생략)

<table>
<tr><td rowspan="6">World</td><td rowspan="3">self.bodies</td><td colspan="2">core/body.py</td><td colspan="2">core/shape.py</td></tr>
<tr><td rowspan="2">Body</td><td>RigidBody</td><td rowspan="2">Shape</td><td>Sphere</td></tr>
<tr><td>CustomRigidBody</td><td>Box</td></tr>
<tr><td rowspan="2">self.backgr ound</td><td colspan="4">core/background.py</td></tr>
<tr><td colspan="4">Background</td></tr>
<tr><td rowspan="5">self.step</td><td colspan="4">core/inegrator.py</td></tr>
<tr><td colspan="2">rk4_step</td><td colspan="2">f</td></tr>
<tr><td colspan="2">core/collision.py</td><td colspan="2">core/event.py</td></tr>
<tr><td colspan="2">collide</td><td colspan="2" rowspan="2">Event</td></tr>
<tr><td colspan="2">resolve</td></tr>
</table>

프로젝트 내용

물리 엔진의 작동 방식

(1) 세계 정의 → (2) 물체 정의 → (3) 물체 추가 → (4) 반복 적분/충돌처리 루프 → (5) 렌더링/입력 처리

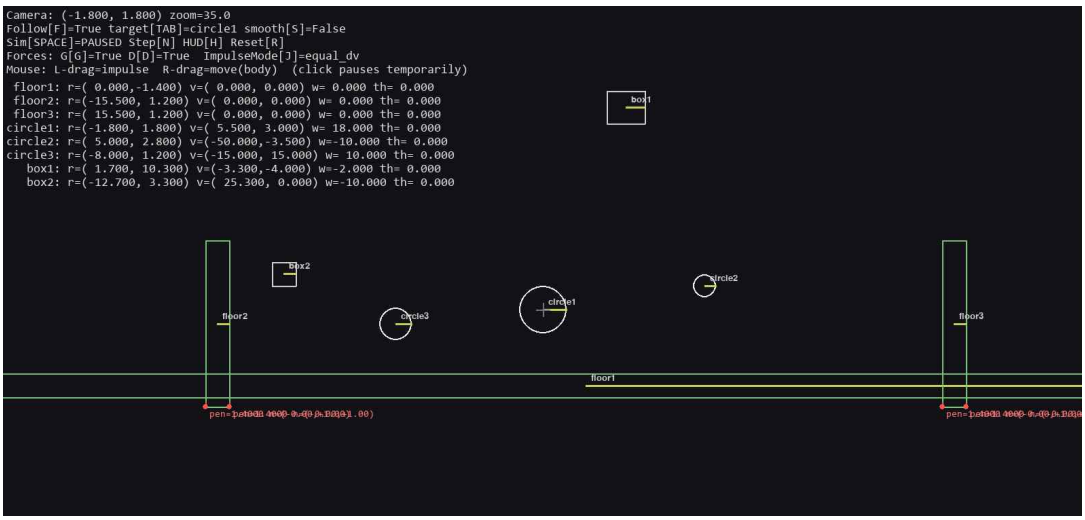
(1) 세계(World) 생성
배경 유체(밀도, 점도)와 회전 감쇠 계수 등을 설정하고, 물체 리스트를 초기화

(2) 물체(Body) 생성
Sphere/Box 등 Shape와 재질 밀도, 초기 위치·속도·회전 상태를 지정하여 RigidBody를 만듦, 이때 질량/관성모멘트/기준면적 등이 자동 계산

(3) World에 물체 추가(add_body)
생성한 물체를 World.bodies에 등록

(4) 시간 간격 dt에 대해 step(dt) 반복 실행(메인 루프)
(a) 적분: 정적 물체(바닥이나 벽 is_state=True 등)를 제외하고, 각 물체에 대해 힘(중력 / 부력 / 항력 / 양력)과 토크를 계산하여 상태 미분방정식을 구성한 뒤, RK4로 다음 상태를 계산해 갱신
(b) 충돌 처리: 모든 물체 쌍에 대해 충돌을 감지하고, 침투 깊이, 법선, 접촉점을 기반으로 충돌 응답(반발/마찰) 및 위치 보정을 반복하여 안정적으로 분리
(c) 제약 유지: z이동 및 x,y축 회전을 제거하여 평면 운동 + z축 회전만 남도록 강제

(5) 렌더링/입력 처리(pygame)
매 프레임마다 월드 좌표를 화면 좌표로 변환해 도형을 그리고, 키/마우스 입력(일시정지, 카메라 이동/줌, 물체 선택 등)을 처리



F : 따라가기 on/off	N : 다음 스텝 실행	G : 중력 및 부력 on/off
TAB : 따라가는 대상 변경	H : HUD 끄기	D : 항력 on/off
SPACE : 일시정지	R : 다시 시작	물체 좌클릭: 속도 추가

충돌 처리 이론 및 구현 방식

충돌이 발생하면 두 물체(body_a, body_b)에 대해 접촉점 목록, 충돌 법선, 침투 깊이을 하나의 이벤트 클래스(core.event.Event)로 묶어 저장한다. 이후 충돌 응답 단계는 이 이벤트를 입력으로 받아 속도, 각속도 및 위치 보정을 수행한다.

(충돌 검출)

본 엔진은 모든 물체 쌍을 검사해 충돌을 탐지하고, 충돌 시 Event를 생성한다. 충돌 검출의 반환 결과는 최소 침투 방향의 법선, 침투 깊이, 실제 접촉점(1~2개)이다. 육면체는 충돌 면을 기준으로 접촉점을 선별해 접촉점 리스트에 넣는다.

(충돌 응답)

충돌 응답은 충돌 직후 속도의 변화를 충격량 J로 처리한다. 각 접촉점 c에서 상대속도를

$$v_{rel} = (v_B + \omega_B \times r_B) - (v_A + \omega_A \times r_a)$$

로 두고, 법선 성분 $v_n = \boldsymbol{v_{rel}} \cdot \boldsymbol{n}$이 접근 중($v_n < 0$)일 때만 반발을 적용한다. 이때 반발계수 e는 두 물체의 값을 사용해 결정하며, 충격량의 크기 j는 질량과 관성모멘트를 포함한 분모(유효질량)으로 나누어 계산한다. 결과적으로 병진 속도와 각속도에 모두 충격이 반영된다.

(마찰 처리)

접선 방향 $\boldsymbol{t}$를 잡아 마찰 충격 j_t를 계산한다. 이때 마찰계수 μ를 이용해 $|j_t| \le \mu j$로 클램핑하여 쿨롱 마찰 한계를 반영한다. 이를 통해 충돌 후 물체에 마찰력이 작용한다.

(반복 위치 보정)

한 프레임에서 충돌을 한 번만 풀면 다중 접촉(연쇄 충돌, 박스가 바닥에 닿는 상황 등)에서 오차가 커지므로, World.step에서 solver_iterations(대략 8~20) 만큼 여러 번 충돌 검출, 응답을 반복하여 점진적으로 해를 수렴시키는 방식을 사용한다.

물리 엔진의 원리(이론)

강체의 운동은 크게 병진 운동과 회전 운동으로 분해하여 나타낼 수 있다. 병진 운동은 질량중심의 위치 벡터 $\boldsymbol{r}$과 속도 벡터 $\boldsymbol{v}$의 변화로 표현되며, 뉴턴의 제2법칙에 의해 다음과 같이 주어진다.

$$m\frac{dt}{dv} = \sum F\ , \quad \frac{dt}{dr} = v$$

여기서 m은 물체 질량, $\sum \boldsymbol{F}$는 물체에 작용하는 모든 외력의 합이다. 본 프로젝트에서는 유체 속 운동을 고려하므로 외력 $\sum \boldsymbol{F}$를 중력, 부력, 항력, 양력(마그누스 힘) 및 충돌로 인한 마찰력으로 구성하였다. 특히 항력은 속도 제곱에 비례하는 표준형 모델을 사용하여

$$F_D = -\frac{1}{2}\rho_f A C_D \| \boldsymbol{v} \| \boldsymbol{v}$$

로 두며, A는 운동 방향에 대한 투영 단면적이고 C_D는 레이놀즈 수에 따라 변하는 항력 계수이다. 회전하는 물체는 자세(회전각)에 따라 투영 단면적이 달라지므로, 자세와 속도 방향을 이용해 A를 투영하여 계산하도록 구성하였다. 이는 동일 속도에서도 자세가 바뀌면 항력이 달라지는 현상을 반영하기 위함이다.

회전 운동은 물체의 각속도 $\boldsymbol{\omega}$와 회전각 θ의 변화로 표현된다. 강체에 대해

$$\boldsymbol{I}\frac{d\boldsymbol{\omega}}{dt} = \sum \boldsymbol{\tau}\ , \ \frac{d\theta}{dt} = \omega$$

가 성립한다. 여기서 $\boldsymbol{I}$는 관성모멘트(또는 관성텐서), $\sum \boldsymbol{\tau}$는 외부 토크의 합이다. 개발한 엔진은 계산 효율과 시각화를 위해 평면 운동 + z축 회전만 허용하는 (3D 연산 -〉 2D 렌더링)하므로, 실제 계산에서는 $\omega = \omega_z$, $I = I_{zz}$만 사용하여

$$I_{zz}\frac{d\omega}{dt} = \tau_z\ , \ \frac{d\theta}{dt} = \omega_z$$

로 단순화된다. 이때 공기(또는 물) 속 회전에 의해 각속도가 감쇠하는 효과는 회전 감쇠 토크로 근사하며, 고속에서 비선형 감쇠를 반영하기 위해 $\tau_z \propto -C_{rot}|\omega_z|\omega_z$ 형태의 모델을 사용하였다. 이는 각속도가 클수록 감쇠가 더 강해지는 경향을 반영한다.

수치적분(RK4)

위의 병진·회전 방정식은 시간에 따른 상태 변화 미분방정식이므로, 시뮬레이션에서는 상태벡터 $y(t)$를 정의하여 1차 미분방정식 형태로 정리한 뒤 수치적으로 적분한다. 예를 들어 본 프로젝트의 핵심 상태는 $\boldsymbol{r}, \boldsymbol{v}, \omega_z, \theta$로 구성되며,

$$\frac{d}{dt}\begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{v} \\ \omega_z \\ \theta \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \frac{1}{m}\sum \boldsymbol{F} \\ \frac{1}{I_{zz}}\tau_z \\ w_z \end{bmatrix}$$

처럼 정리할 수 있다. 이때 항력, 양력, 충돌 등으로 인해 $\sum \boldsymbol{F}$가 속도 및 자세에 비선형으로 의존하므로, 해석해를 구하기보다 시간 간격 dt마다 수치적분으로 다음 상태를 계산하는 방식이 적절하다. 본 프로젝트에서는 오일러 방법 대비 오차가 작고 안정성이 높은 4차 룽게-쿠타 방법(RK4)을 적용하여, 한 스텝에서 여러 번 기울기(미분값)를 샘플링한 뒤 가중 평균으로 상태를 갱신한다. 이로써 동일한 dt에서도 빠르게 변하는 비선형 운동을 비교적 안정적으로 추적할 수 있다.

프로젝트 결과

초기 버전	https://youtu.be/5TVlUrs_HKs?si=-ZtLtAI6zEWRcOQs
공기 속	https://youtu.be/VJ8VrAnBY-E?si=6x1q1H0poEioDM37
공기 속, 중력/부력/항력X	https://youtu.be/lIpDjDhGlJA?si=AhzIzUFuTtk7ZWHK
물 속	https://youtu.be/8F-jhoDHsko?si=rFNg9kERJCa9xNU2 밀도가 물보다 작은 circle1과 box2는 9초~10초, 17초~20초에서 볼 수 있듯이 위로 뜨며, 밀도가 큰 다른 세 물체는 가라앉는다.
물 속 중력 부력 항력X	https://youtu.be/lIpDjDhGlJA?si=T165QtxwfZ-mj0eA
물 속 물체의 밀도에 따른 부력 효과	https://youtu.be/fjNrynah52Y?si=rZrEg-76pUivcuo0

위와 같이 완성하였다.
config.py 파일에서 다양한 물질의 밀도를 정의하였기 때문에
물체의 점성 계수만 따로 추가하면 다양한 재질의 물체나, 배경등을 적용할 수 있다. 또한 shape에서 유체 클래스를 따로 추가한다면 수영장처럼 일부 영역에 물이 차있는 형태도 만들 수 있을 것이다.

반성 및 심화 프로젝트 주제 설정

가장 반성할 점은 충돌 처리 과정에서 매우 중요한 광역 탐지부분을 만들지 못했다. 그 원인은 적분기와 객체지향 구조에 너무 치중하느라 충돌처리에 제대로 집중하지 못했다고 생각한다. 자세히 말하자면, 본 엔진의 가장 큰 문제는 물체를 검사할 때 모든 쌍을 검사한다. 따라서 물체가 늘어나면 시간복잡도가 $O(N^2)$이므로 검사 비용이 매우 커진다. 따라서 공간 분할(그리드/쿼드트리)이나 AABB 기반 broad-phase를 도입하여 충돌 후보군을 먼저 줄이고, 그 이후에 정밀 충돌을 수행하는 구조로 개선할 필요가 있다.

또한 시간 간격 dt가 커질 때(물체의 속도가 매우 빠를 때) 충돌 침투 정도가 증가하거나 에너지가 비정상적으로 커지거나 아예 통과하는 현상이 있다. 고정 dt보다는 물체의 속도에 따라 가변적인 dt(추가적인 서브 스텝으로 시간을 더 쪼갬)나 충돌 보정의 횟수를 유동적으로 조절하는 시스템을 만들어야 한다.

마지막으로 렌더링과 입력 처리 부분은 현재 기본 기능 위주로 구성되어 있으므로, 디버그 시각화(충돌 법선/접촉점 표시, 속도 벡터 표시), 로그 저장, 재현 가능한 시뮬레이션 시드 관리 등을 추가하여 검증 가능한 엔진으로 발전시키는 것을 목표로 한다. 또한 API의 형식으로 웹 등의 호스팅하는 방법으로도 일반적인 서버형 물리엔진을 구현하는 방법에 대해 알아보고자 한다.