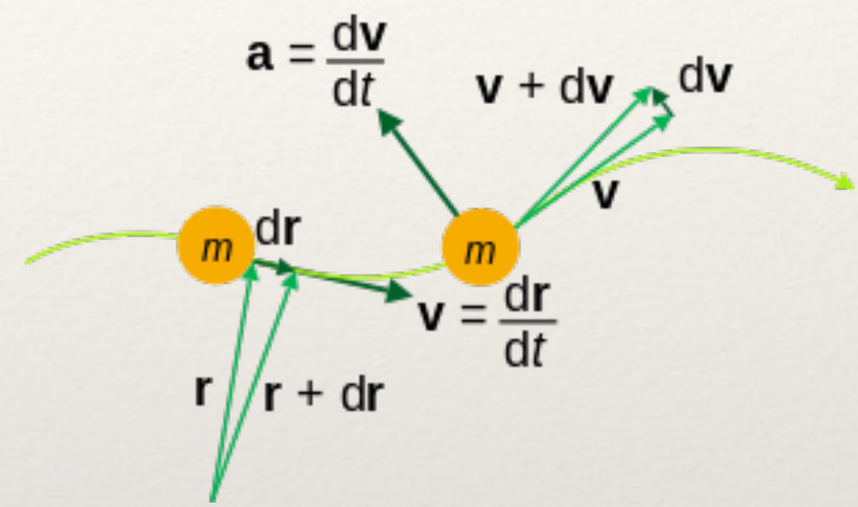
물리기반 모델링

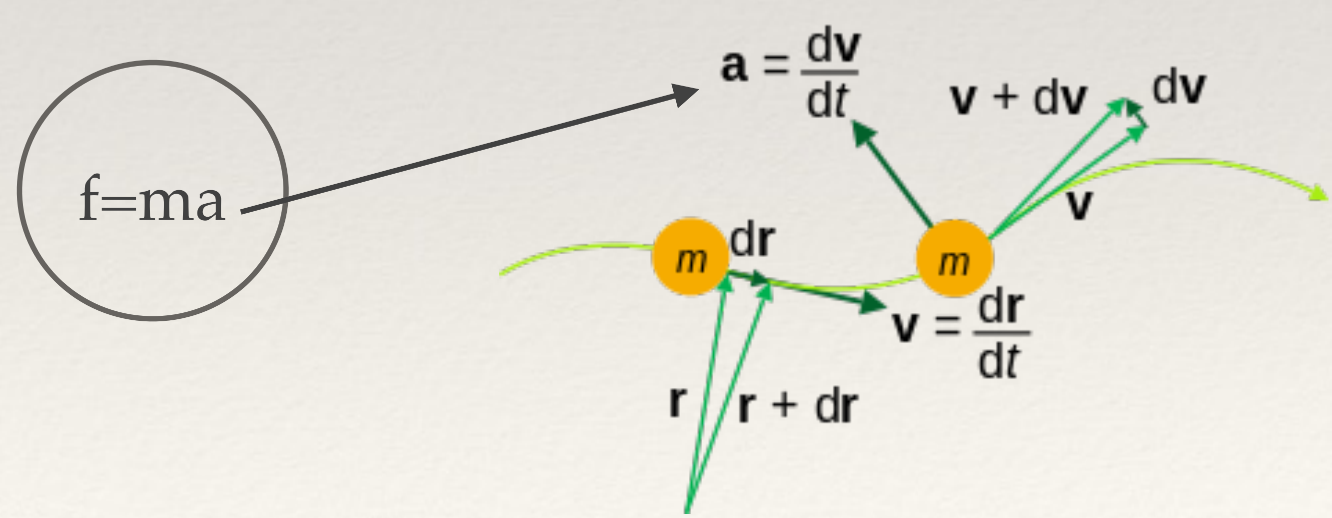
1.2 운동학

동명대학교
강영민

운동학

- 운동학과 동역학
  - 운동학
    - 힘이 고려되지 않음
  - 위치, 속도, 가속이 시간의 함수로 다뤄짐
    - x(t), v(t), a(t)
- 동역학
  - 힘이 가장 중요한 역할
    - 현재 상태의 힘 f(t)을 계산
    - 가속 계산 a(t) = f(t) / m
    - 속도 갱신 v(t+dt) += a(t)dt
    - 위치 갱신 x(t+dt) += v(t+dt)dt





정역학, 운동학, 동역학, 역학

- 정역학(statics)
  - 평형 상태와 힘의 관계를 연구
- 동역학(kinetics)
  - 운동과 힘의 관계를 연구
- 운동학(kinematics)
  - 관찰된 동작을 이 동작을 유발하는 힘에 대한 고려 없이 연구

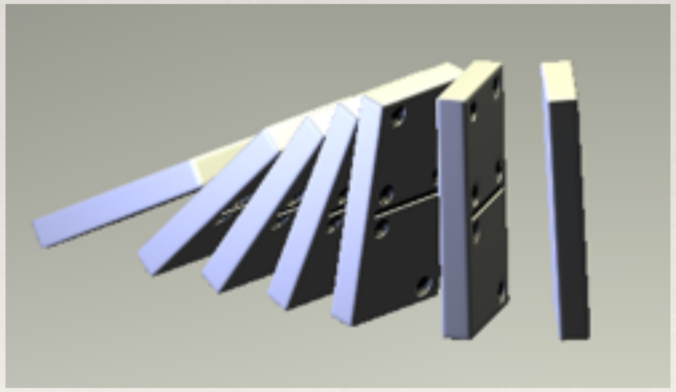
classical mechanics

- 역학(dynamics)
  - 정역학 + 동역학

입자와 강체

- 입자
  - 질량을 가진 아주 아주 작은 객체
  - 부피는 무시할 수 있음
    - 예: 로켓 탄도 분석
      - 로켓의 부피는 무시할 수 있음
      - 로켓도 입자로 간주할 수 있음

- 강체 (이상적인 고체)
  - 질량과 부피를 가진 객체
  - 어떤 경우에도 모양이 변하지 않음
  - 강체의 유효한 애니메이션 = 회전과 이동



속도

- 속도(velocity)
  - 벡터
  - 속력(speed): 속도의 크기
  - 속도 = (속력, 방향)
  - 속도의 방향
    - 이동하는 방향
  - 속도의 크기
    - 시간에 대해 이동하는 거리의 비

$$\mathbf{v} = \frac{\Delta \mathbf{s}}{\Delta t}$$

ratio of displacement to time interval

순간 속도

- 시간은 흐른 시간에 대한 이동의 비
  - 시간 간격을 줄이면...
  - 측정하는 순간에 대해 더 정확한 속도를 얻게 됨
  - 시간 간격이 0에 접근할 때 (= 위치를 시간에 대해 미분)
    - 이를 순간 속도라고 함

$$\mathbf{v} = \lim_{\Delta t \to 0} \frac{\Delta \mathbf{s}}{\Delta t} = \frac{d\mathbf{s}}{dt}$$

변위 (displacement)

- 속도의 적분

$$\int \mathbf{v} dt = \int d\mathbf{s}$$

$$\mathbf{v} = \lim_{\Delta t \to 0} \frac{\Delta \mathbf{s}}{\Delta t} = \frac{d\mathbf{s}}{dt}$$

$$\mathbf{v} dt = d\mathbf{s}$$

  - t1에서 t2 시간 간격 동안의 적분

$$\int_{t_1}^{t_2} \mathbf{v} dt = \int_{s(t_1)}^{s(t_2)} d\mathbf{s}$$

  - 해당 시간 동안 이루어지는 이동량

$$\int_{t_1}^{t_2} \mathbf{v} dt = \mathbf{s}(t_2) - \mathbf{s}(t_1) = \Delta \mathbf{s}$$

- 만약 속도를 안다면,
  - 미래에 이 입자가 어디에 있을지를 알 수 있음

가속

- 평균 가속
  - 주어진 시간 간격에 대해 변화한 속도의 비

$$\mathbf{a} = \Delta\mathbf{v}/\Delta t$$

- 순간 가속도

$$\mathbf{a} = \lim_{\Delta t \to 0} \Delta\mathbf{v}/\Delta t = d\mathbf{v}/dt$$

- 가속도의 적분

$$\mathbf{a}dt = d\mathbf{v}$$

  - 속도의 변화

$$\int_{t_1}^{t_2} \mathbf{a}dt = \int_{\mathbf{v}(t_1)}^{\mathbf{v}(t_2)} d\mathbf{v} = \Delta\mathbf{v}$$

등가속 운동

- 운동학의 단순한 문제
  - 등가속운동의 예: 중력
  - 중력 가속도
    - 크기: 9.81 m/s^2
    - 방향: 아래쪽 (0,-1,0)
- 가속도가 상수이므로 쉽게 적분할 수 있다.
  - 이 적분을 통해
    - 매 순간 속도의 변화를 알 수 있으며,
    - 매 순간 속도를 계산할 수 있다.

중력 가속 문제

- 중력 가속도: g
  - 속도의 변화와 가속의 적분 사이의 관계

$$\int_{\mathbf{v}(t_1)}^{\mathbf{v}(t_2)} d\mathbf{v} = \int_{t_1}^{t_2} \mathbf{g}dt$$

$$\mathbf{v}_2 - \mathbf{v}_1 = \mathbf{g}(t_2 - t_1)$$

  - t_1에서의 상태를 안다면
  - 쉽게 t_2에서의 상태를 알 수 있다.

$$\mathbf{v}_2 = \mathbf{g}t_2 - \mathbf{g}t_1 + \mathbf{v}_1$$

    - t_1=0이고 t_2=t라면…

$$\mathbf{v}_2 = \mathbf{v}_1 + \mathbf{g}t$$

변위에 대한 함수로의 속도

$$\mathbf{v}\frac{d\mathbf{v}}{dt} = \frac{d\mathbf{s}}{dt}\mathbf{a}$$

- 또 다른 미분 방정식

$$\mathbf{v}d\mathbf{v} = \mathbf{a}d\mathbf{s}$$

- 적분…

$$\int_{\mathbf{v}_1}^{\mathbf{v}_2} \mathbf{v}d\mathbf{v} = \int_{\mathbf{s}_1}^{\mathbf{s}_2} \mathbf{a}d\mathbf{s}$$

$$\frac{1}{2}\mathbf{v}^2\Big|_{\mathbf{v}_1}^{\mathbf{v}_2} = \mathbf{a}\mathbf{s}\Big|_{\mathbf{s}_1}^{\mathbf{s}_2} = \mathbf{g}\mathbf{s}\Big|_{\mathbf{s}_1}^{\mathbf{s}_2}$$

- 속도와 변위 사이의 관계

$$\frac{1}{2}({\mathbf{v}_2}^2 - {\mathbf{v}_1}^2) = \mathbf{g}(\mathbf{s}_2 - \mathbf{s}_1)$$

$${\mathbf{v}_2}^2 = 2\mathbf{g}(\mathbf{s}_2 - \mathbf{s}_1) + {\mathbf{v}_1}^2$$

위치

- 속도와 변위

$$\mathbf{v}dt = d\mathbf{s}$$
$$(\mathbf{v}_1 + \mathbf{g}t)dt = d\mathbf{s}$$

- 적분

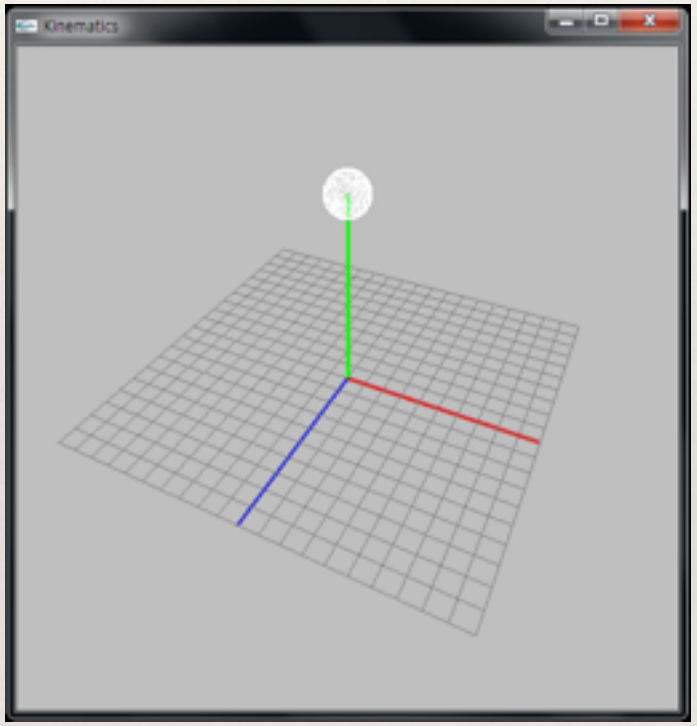
$$\int_0^t (\mathbf{v}_1 + \mathbf{g}t)dt = \int_{\mathbf{s}_1}^{\mathbf{s}_2} d\mathbf{s}$$
$$v_1 t + \frac{1}{2}\mathbf{g}t^2 = \mathbf{s}_2 - \mathbf{s}_1$$

- 시간 t에서의 위치

$$\mathbf{s}_2 = v_1 t + \frac{1}{2}\mathbf{g}t^2 + \mathbf{s}_1$$

운동학 시뮬레이션

```
#include "KinematicsSimulator.h"

CKinematicSimulator::CKinematicSimulator() : CSimulator() {}


void CKinematicSimulator::init() {
    initialLoc.set(0,1,0);
    initialVel.set(0,3,0);
    gravity.set(0.0, -9.8, 0.0);
    currentLoc = initialLoc;
    particle.setPosition(currentLoc[0], currentLoc[1], currentLoc[2]);
    particle.setRadius(0.1);
}

void CKinematicSimulator::doBeforeSimulation(double dt, double currentTime) {

}

void CKinematicSimulator::doSimulation(double dt, double currentTime) {
    currentLoc = initialLoc
        + currentTime*initialVel
        + (0.5 * currentTime * currentTime)  * gravity ;
    particle.setPosition(currentLoc[0], currentLoc[1], currentLoc[2]);
    particle.drawWithGL();
}


void CKinematicSimulator::doAfterSimulation(double dt, double currentTime) {

}
```



https://github.com/dknife/201501Lec_GamePhysics/tree/master/GPcode01_03_KinematicParticleExplosion

운동학을 통한 입자 폭발 효과

- KinematicSimulator.cpp

```
#include "KinematicsSimulator.h"

CKinematicSimulator::CKinematicSimulator() : CSimulator() {}
void CKinematicSimulator::init() {

    for(int i=0;i<NUMPARTS;i++)particle[i].randomInit();
}

void CKinematicSimulator::doBeforeSimulation(double dt, double currentTime) { }
void CKinematicSimulator::doSimulation(double dt, double currentTime) {
    for(int i=0;i<NUMPARTS;i++){
        particle[i].simulate(dt, currentTime);
        particle[i].drawWithGL(POINT_DRAW);
    }
}
void CKinematicSimulator::doAfterSimulation(double dt, double currentTime) { }
```

운동학을 이용한 입자 폭발 효과

- Particle.cpp

```
void CParticle::randomInit() {
    double speed = rand()%10000 / 10000.0 + 1.0;
    double theta = 2.0*3.141592 * (rand()%10000 / 10000.0);
    double phi =   2.0*3.141592 * (rand()%10000 / 10000.0);
    double vx,vy,vz;  // spherical coord to cartesian coord
    vy = speed*cos(phi)+2.0;
    vx = speed*cos(theta)*sin(phi);
    vz = speed*sin(theta)*sin(phi);
    initialLoc.set(0,1,0);
    initialVel.set(vx, vy, vz);
    gravity.set(0.0, -9.8, 0.0);
    radius = 0.01;
    currentLoc = initialLoc;
}
void CParticle::simulate(double dt, double et) {
    currentLoc = initialLoc + et*initialVel + (0.5 * et * et)  * gravity ;
    setPosition(currentLoc[0], currentLoc[1], currentLoc[2]);
}
```

결과