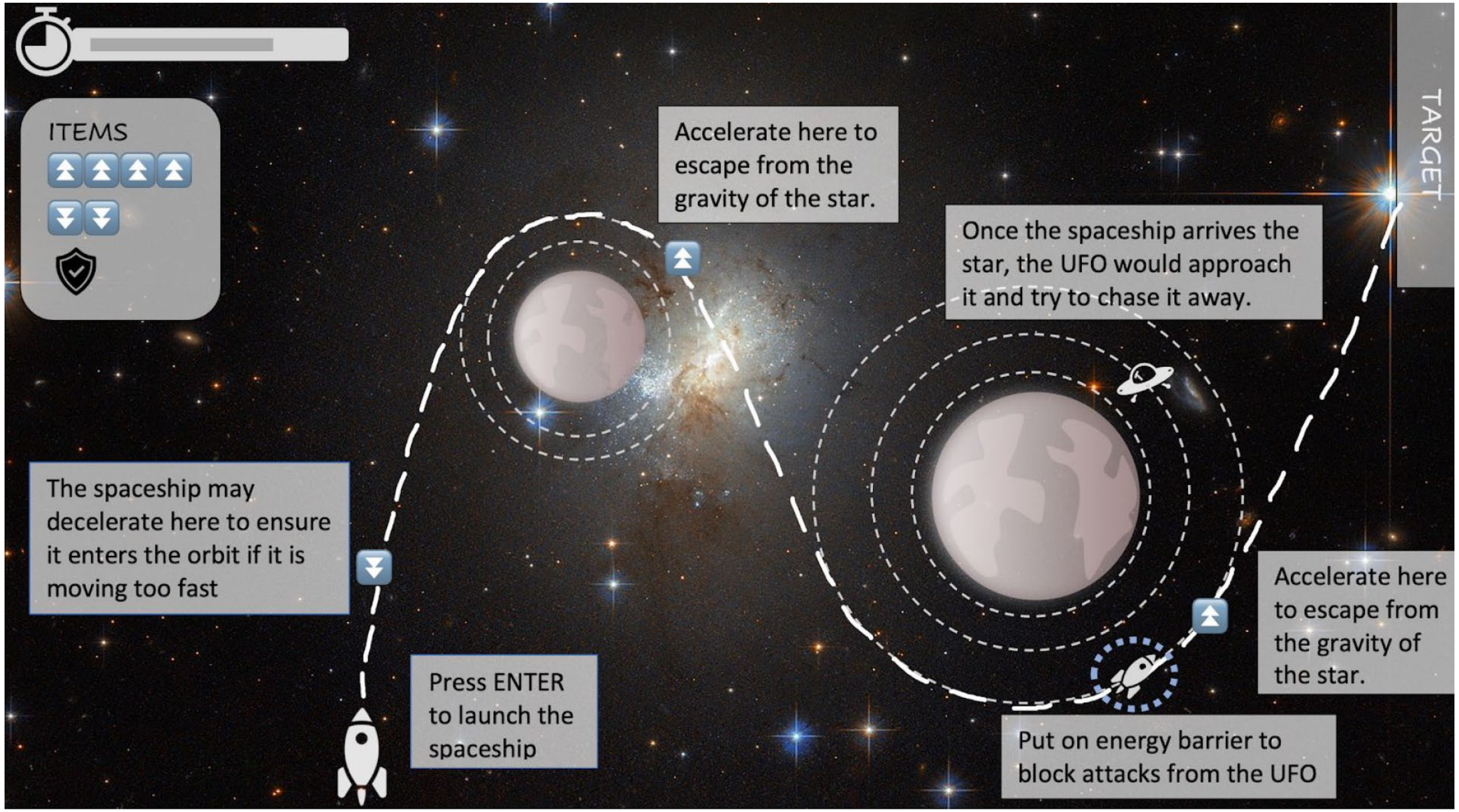
Helge Rhodin

CPSC 427
Video Game Programming

Physical Simulation



Feature: Level Loading with JSON

Libraries:

- ***https://sourceforge.net/projects/libjson/***
- ***https://github.com/nlohmann/jso***
- ***others?***

Loading Entities and Components

- Outer list of entities
- Inner list of components
- Create a factory that instantiates each component type
- Equip components with toJSON(…) and fromJSON(…) functions

```
"entities": [
  {
    "position": {
      "x": -1.7193701,
      "y": -0.09165986
    },
    "velocity": {
      "x": 0,
      "y": 0
    },
    "color": {
      "x": 0.453125,
      "y": 0.453125,
      "z": 0.453125
    },
    "type": "Water Animal"
  },
  {
    "position": {
      "x": 2.2221813,
      "y": -1.2671415
    },
    "velocity": {
      "x": 0,
      "y": 1
    },
    "radius": 0.9300000000000006,
    "color": {
      "x": 0.40625,
      "y": 0.40625,
      "z": 0.40625
    },
    "type": "Land Animal"
  }
]
```

Factory from JSON

Factory:

```
void ComponentfromJson(Entity e, JsonObject json)
{
  if(str1.compare("Motion") != 0) {
     Motion& motion = Motion::fromJson(json);
     registry->insert(e, motion);
  }
  else if(str1.compare("Salmon") != 0)
     Motion& component = Motion::fromJson(json);
     registry->insert(e, component);
  }
  …
}
```

Issues?

Component from JSON

Component to/from:

```
class Vector2D
{
    float x,y;
    public:
    JsonObject toJson()
    {
       JsonObject json = Json.object();
       json.add("x", x);
       json.add("y", y);
       return json;
    }

    static Vector2D fromJson(JsonObject json)
    {
       double x = json.getFloat("x", 0.0f);
       double y = json.getFloat("y", 0.0f);
       return Vector2D(x,y);
    }
}
```

Setup

@Helge: Pressed record?

@Class: Logged into iClicker cloud?

Overview

1. Equation of Motion

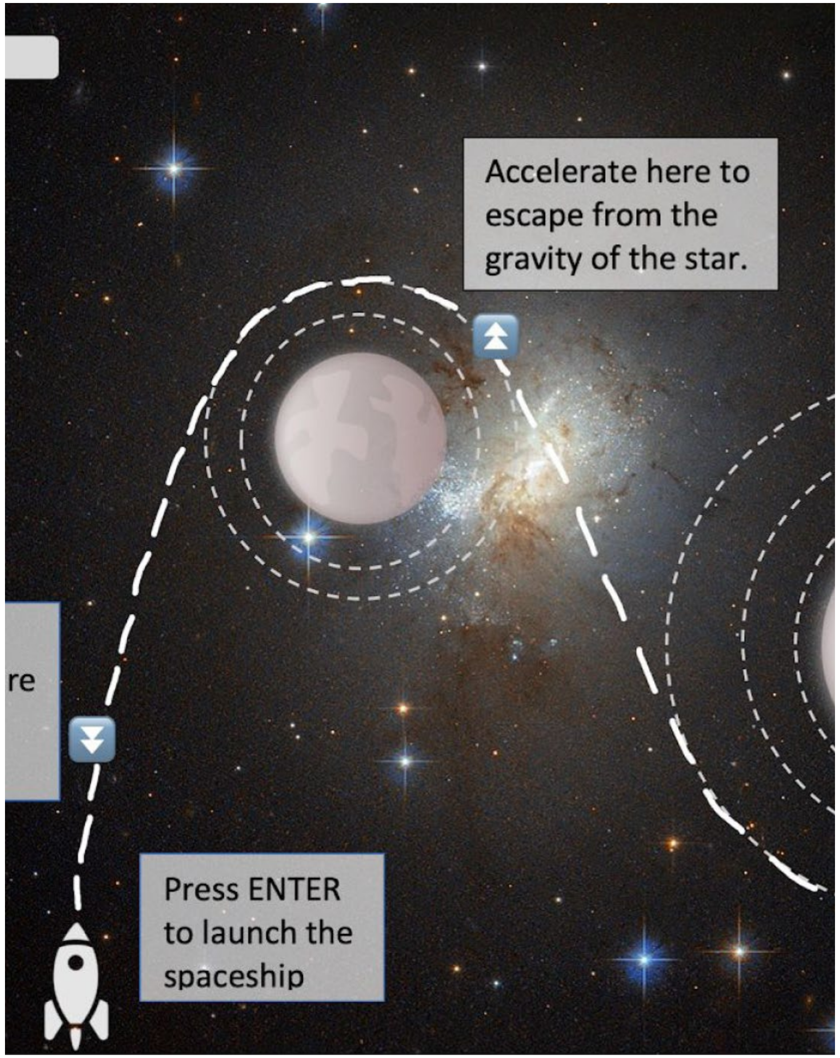
- Examples
- Ordinary Differentiable Equations (ODE)
- Solving ODEs

2. Collision and Reaction Forces

Physics

Learning goals:

- ***Connect your theoretical math knowledge to applications***
- ***Properly simulate object motion and their interaction in your game***



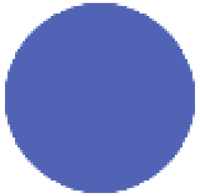
Recap: Basic Particle Simulation (first try)

How to compute the change in velocity?

$$\boldsymbol{d_t} = \boldsymbol{t_{i+1}} - \boldsymbol{t_i}$$

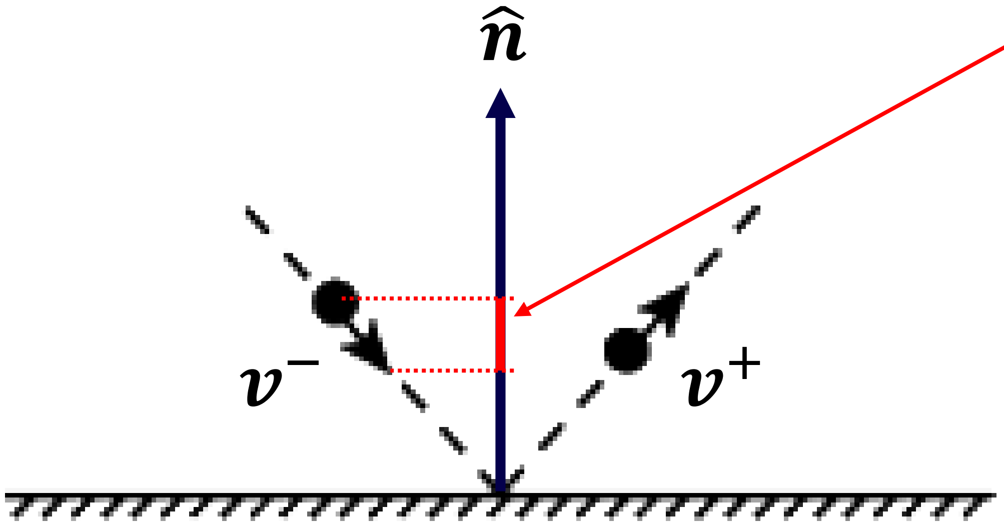
$$\vec{\boldsymbol{v}}_{\boldsymbol{i+1}} = \vec{\boldsymbol{v}}_{\boldsymbol{i}} + \color{red}{\boldsymbol{\Delta v}}$$

$$\vec{\boldsymbol{p}}_{\boldsymbol{i+1}} = \vec{\boldsymbol{p}}(\boldsymbol{t_i}) + \vec{\boldsymbol{v}}_{\boldsymbol{i}}\boldsymbol{d_t}$$



Recap: Particle-Plane Collision

- *In direction of normal*



Velocity along normal (v projected on normal by the dot product)

Frictionless

$$\Delta v = 2(v^- \cdot \hat{n})\hat{n}$$

Apply change along normal (magnitude times direction)

$$v^+ = v^- + \Delta v$$

Loss of energy

$$\Delta v = (1+\epsilon)(v^- \cdot \hat{n})\hat{n}$$

Particle-Particle Collisions (spherical objects)

Before collision **After**

Response:

$$v_1^+ = v_1^- - \frac{2m_2}{m_1 + m_2} \frac{\langle v_1^- - v_2^- \rangle \cdot \langle p_1 - p_2 \rangle}{\|p_1 - p_2\|^2} \langle p_1 - p_2 \rangle$$

$$v_2^+ = v_2^- - \frac{2m_1}{m_1 + m_2} \frac{\langle v_2^- - v_1^- \rangle \cdot \langle p_2 - p_1 \rangle}{\|p_2 - p_1\|^2} \langle p_2 - p_1 \rangle$$

- This is in terms of velocity
- Today (and next lecture): derivation via impulse and forces

From Velocities (Δv) to Forces (F) and back

Force relates to mass and acceleration

$$\mathbf{F} = ma$$

A change in velocity related to acceleration over time

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

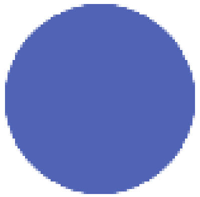
In terms of forces

$$\Delta \mathbf{v} = \Delta t \, \frac{F}{m}$$

Recap: Basic Particle Simulation (first try)

How to compute the change in velocity?

$$d_t = t_{i+1} - t_i$$
$$\vec{v}_{i+1} = \vec{v}_i + \Delta v$$
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}_i d_t$$



Forces are omnipresent

- **Gravity**

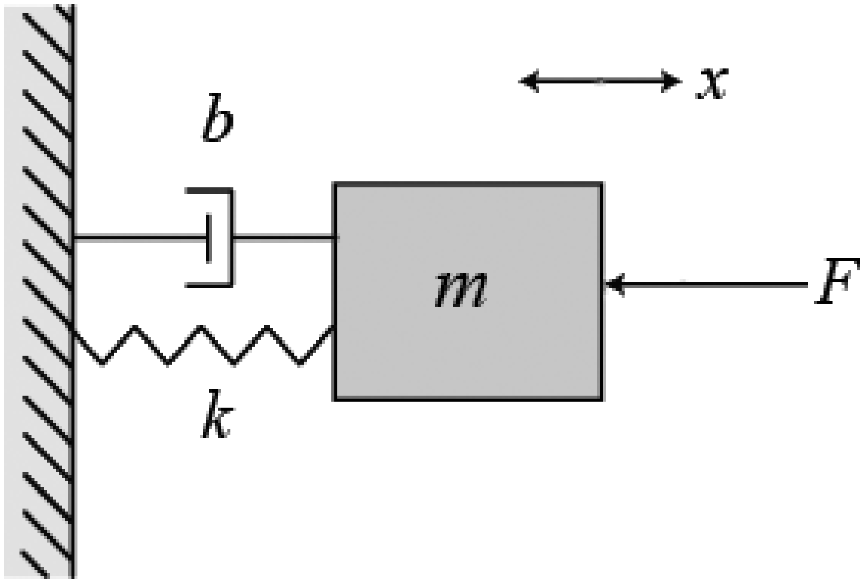
$$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

- **Viscous damping**

$$F = -bv$$

- **Spring & dampers**

$$F = -kx - bv$$



Gravity direction?

Assuming a flat earth:

$$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

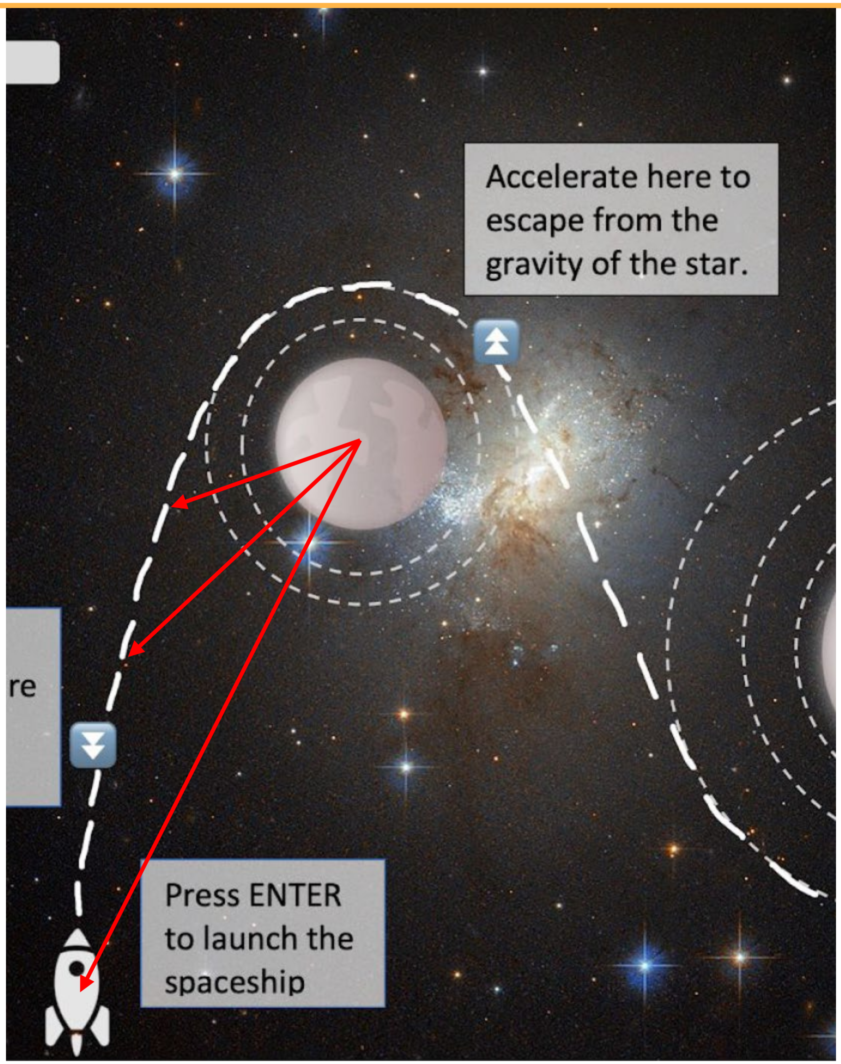
Assuming a spherical earth:

$$F = -mg \begin{bmatrix} a \\ b \end{bmatrix}$$

How to compute the vector (a,b) and g ?

Newton's law of universal gravitation

$$F = G\frac{m_1 m_2}{r^2}$$



Multiple forces?

Forces add up (and cancel):

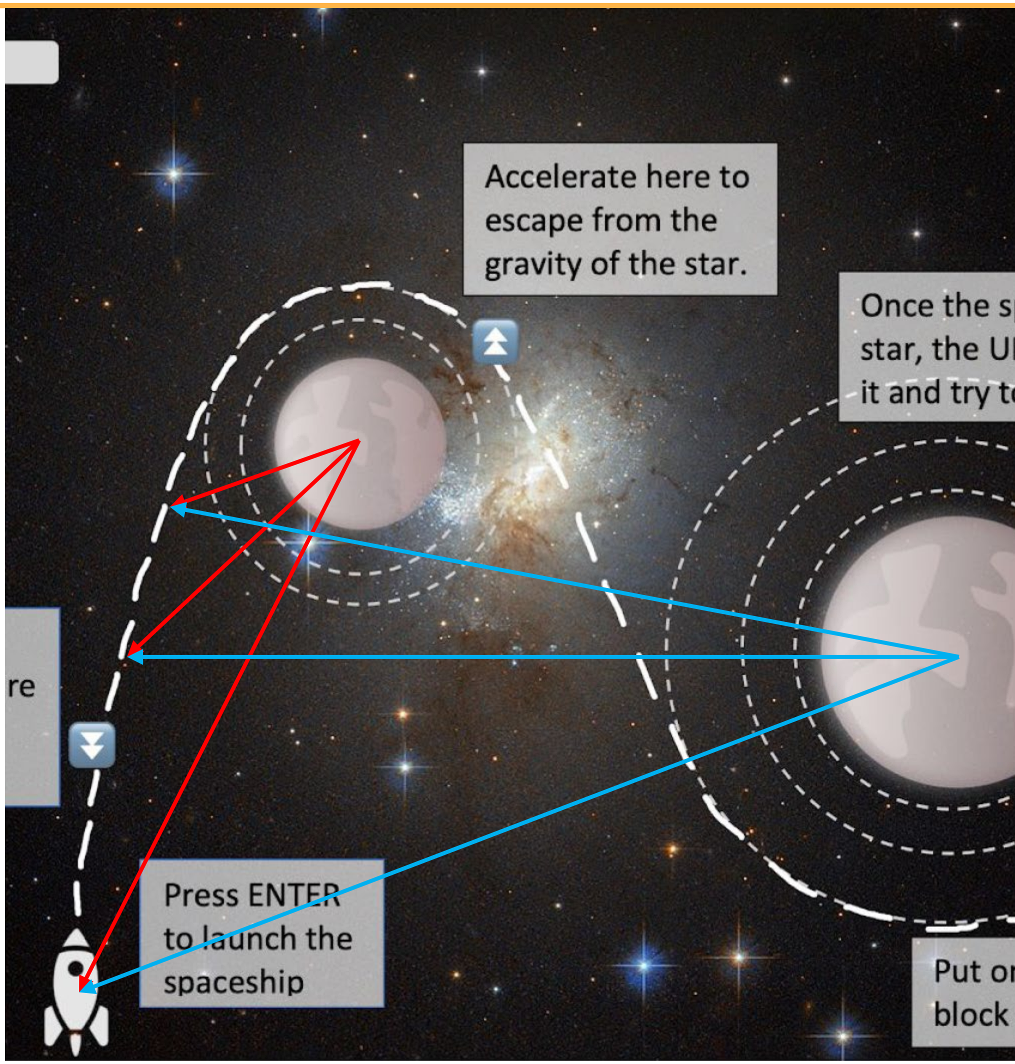
$$F = -mg_1 \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} - mg_2 \begin{bmatrix} a_2 \\ b_2 \end{bmatrix}$$

- ***This holds for all types of forces!***

- ***Notation you might see:***

$$F = \sum_i F_i = \sum F_i = \sum F$$

$$\vec{F} = F$$



Your game idea does not need forces?

Are you sure?

- ***Particle effects***
- ***Fake forces***
- ***Proxy forces***
  - Simulate crowd behaviour

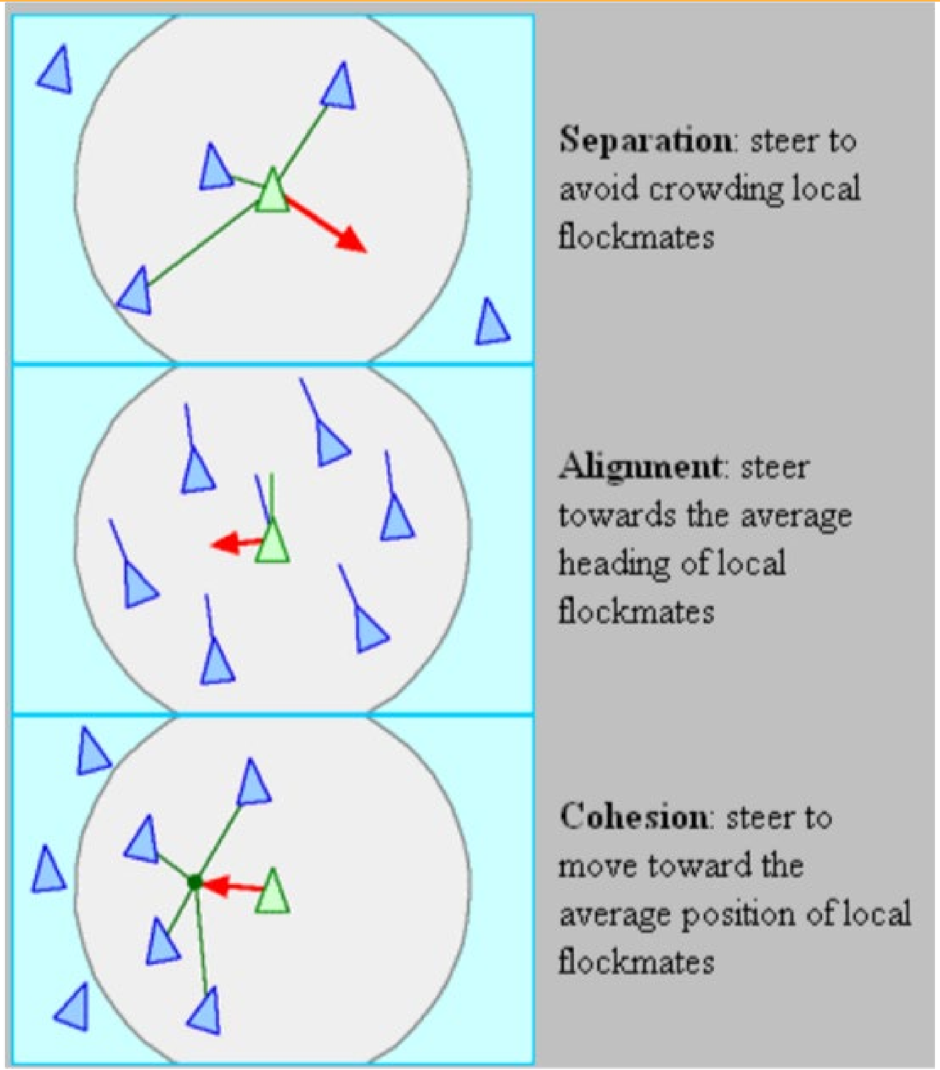
Take it as a chance to connect dry math with a practical application!

Proxy Forces (= fake forces)

- **Behavior forces: [“Boids”, Craig Reynolds, SIGGRAPH 1987]**
- **flocking birds, schooling fish, etc.**

- **Attract to goal location (like gravity)**
  - ***E.g., waypoint determined by shortest path search***
- **Repulsion if close**
- **Align orientation to neighbors**
- **Center to neighbors**

- **Forces add up!**



Simulation Basics

Simulation loop…

1. ***Equations of Motion***
   - sum forces & torques
   - solve for accelerations: $\vec{\boldsymbol{F}} = \boldsymbol{ma}$
2. ***Numerical integration***
   - update positions, velocities
3. ***Collision detection***
4. ***Collision resolution***

What we did so far: Forward Euler

Forces only $\vec{F} = ma$

$$d_t = t_{i+1} - t_i$$
$$\vec{v}_{i+1} = \vec{v}_i + (\vec{F}(t_i)/m)d_t$$
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}_{i+1}d_t$$

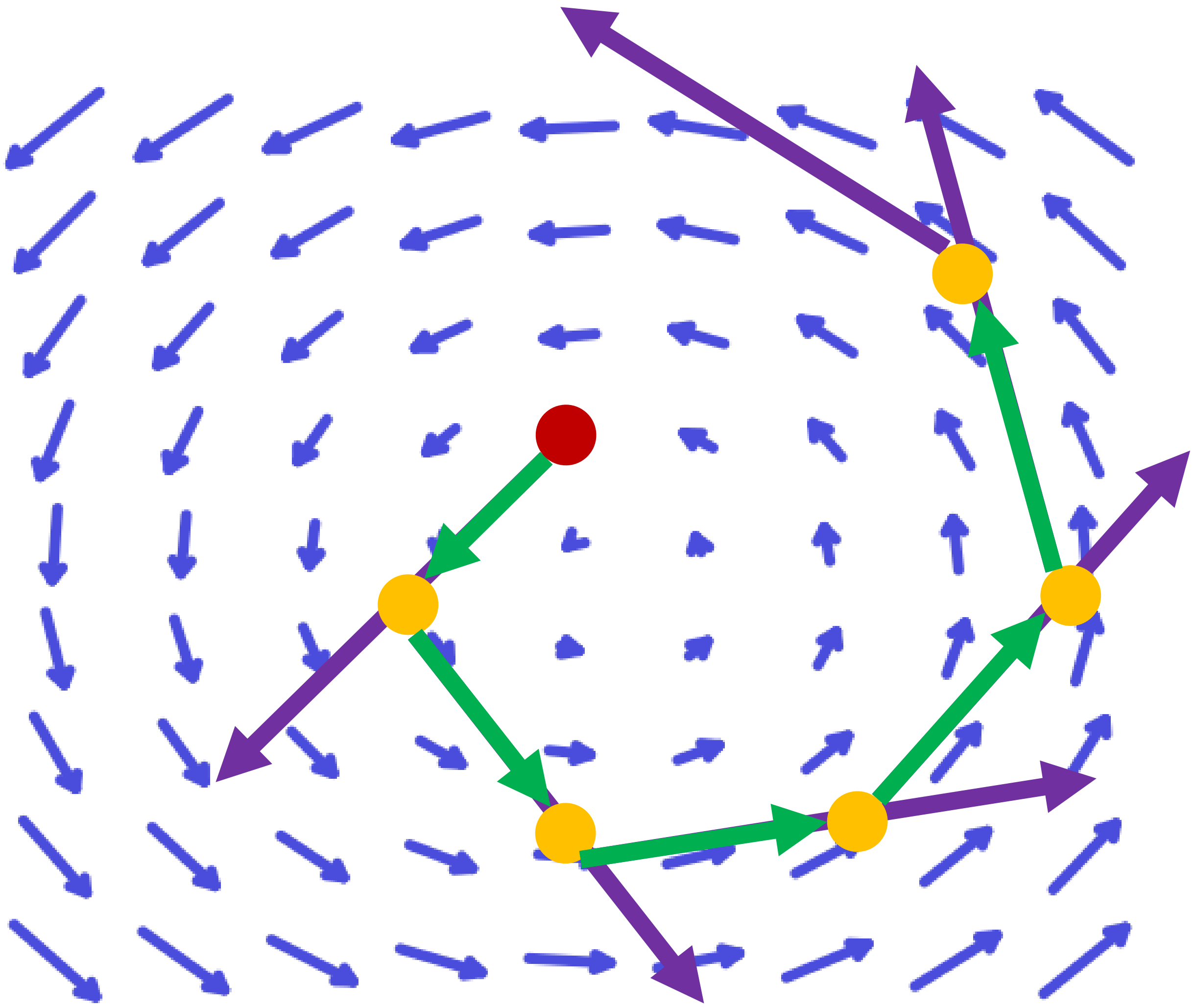
acceleration $= \frac{\partial v}{\partial t}$

get values at time t_{i+1} from values at time t_i

Issues? Alternatives?

How can we discretize this?

Issue: extrapolation

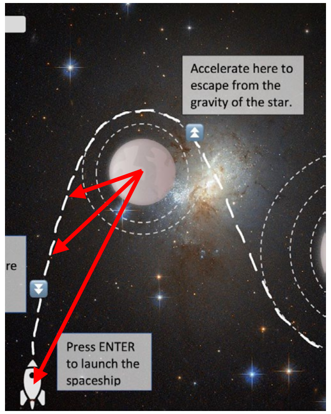


Which forces depend on t?

- **Gravity**

$$F = \begin{bmatrix} 0 \\ -mg \end{bmatrix}$$

$$F = -mg \begin{bmatrix} a \\ b \end{bmatrix}$$

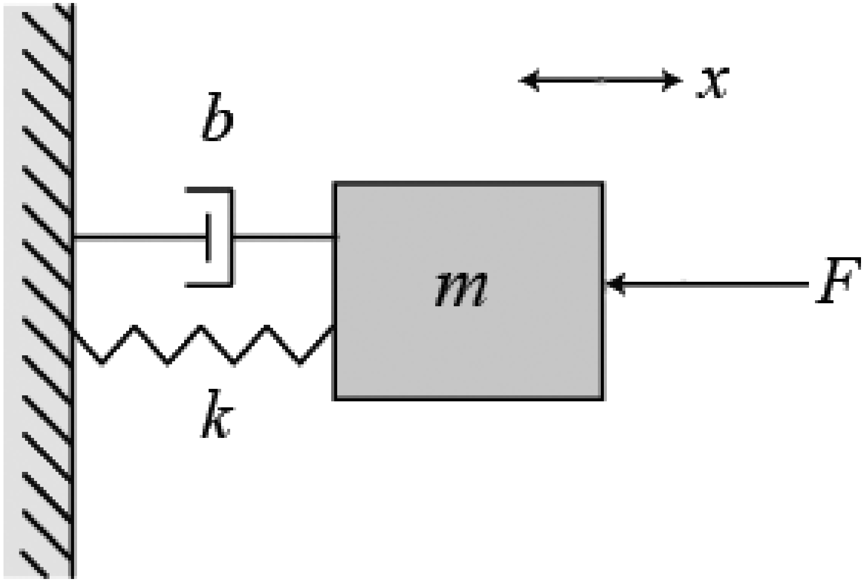


- **Viscous damping**

$$F = -bv$$

- **Spring & dampers**

$$F = -kx - bv$$



Basic Particle Simulation: Small Problem…

$$d_t = t_{i+1} - t_i$$

$$\vec{v}_{i+1} = \vec{v}_i + (\vec{F}(t_{???})/m)d_t$$

$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}_{i+1}d_t$$

Equations of motion describe state (equilibrium)

- **Involves quantities and their derivatives**
  - **-> we need to solve differential equations**

Lets start from scratch

Given:

$$\vec{F} = m\,\frac{\partial^2 x}{\partial t^2}$$

Wait!
There is no position x in this equation?!
Only contains acceleration a!

How to solve such differential equation?

Desired: the position x at time t

Newtonian Physics as First-Order Diff. Eq. (DE)

Second-order DE

$$\vec{F} = m \frac{\partial^2 x}{\partial t^2}$$

acceleration

$= \frac{\partial v}{\partial t}$

First-order DE

$$\frac{\partial}{\partial t}\begin{bmatrix}\vec{x} \\ \vec{v}\end{bmatrix} = \begin{bmatrix}\vec{v} \\ \vec{F}/m\end{bmatrix}$$

velocity

$= \frac{\partial x}{\partial t}$

Higher-order DEs can be turned into a first-order DE with additional variables and equations!

Newtonian Physics as First-Order DE

- Motion of one particle

Second-order DE

$$\vec{F} = m\,\frac{\partial^2 x}{\partial t^2}$$

First-order DE

$$\frac{\partial}{\partial t}\begin{bmatrix}\vec{x}\\ \vec{v}\end{bmatrix} = \begin{bmatrix}\vec{v}\\ \Sigma\vec{F}/m\end{bmatrix}$$

- Motion of many particles

$$\frac{\partial}{\partial t}\begin{bmatrix}\overrightarrow{x_1}\\ \overrightarrow{v_1}\\ \overrightarrow{x_2}\\ \overrightarrow{v_2}\\ \vdots\\ \overrightarrow{x_n}\\ \overrightarrow{v_n}\end{bmatrix} = \begin{bmatrix}\overrightarrow{v_1}\\ \overrightarrow{F_1}/m_1\\ \overrightarrow{v_2}\\ \overrightarrow{F_2}/m_2\\ \vdots\\ \overrightarrow{v_n}\\ \overrightarrow{F_n}/m_n\end{bmatrix}$$

Overview

Different DE solvers

- ***Forward Euler***
  (take current accel. to update vel., current vel. to update pos.)

- ***Midpoint Method & Trapezoid Method***
  (mix current and approximations of future vel. & acc. Estimates)

- ***Backwards Euler***
  (solve for future pos., vel., and accel. jointly)
  - *May require an iterative solver*

Differential Equations

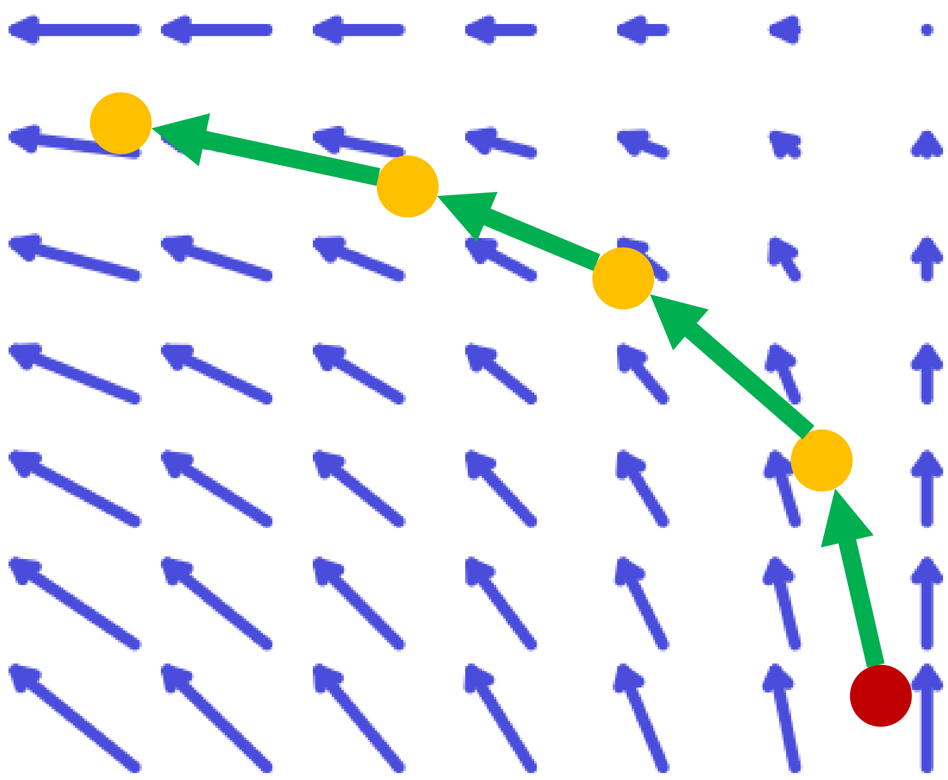
$$\frac{\partial}{\partial t}\vec{X}(t) = f(\vec{X}(t), t)$$

Given that $\vec{X}_0 = \vec{X}(t_0)$

Compute $\vec{X}(t)$ **for** $\mathrm{t} > t_0$

e.g., $\Delta\vec{X}(t) = f(\vec{X}(t), t)\Delta t$

- **Simulation:**
  - ***path through state-space***
  - ***driven by vector field***



DE Numerical Integration: Explicit (Forward) Euler

$$\frac{\partial}{\partial t}\vec{X}(t) = f(\vec{X}(t), t)$$

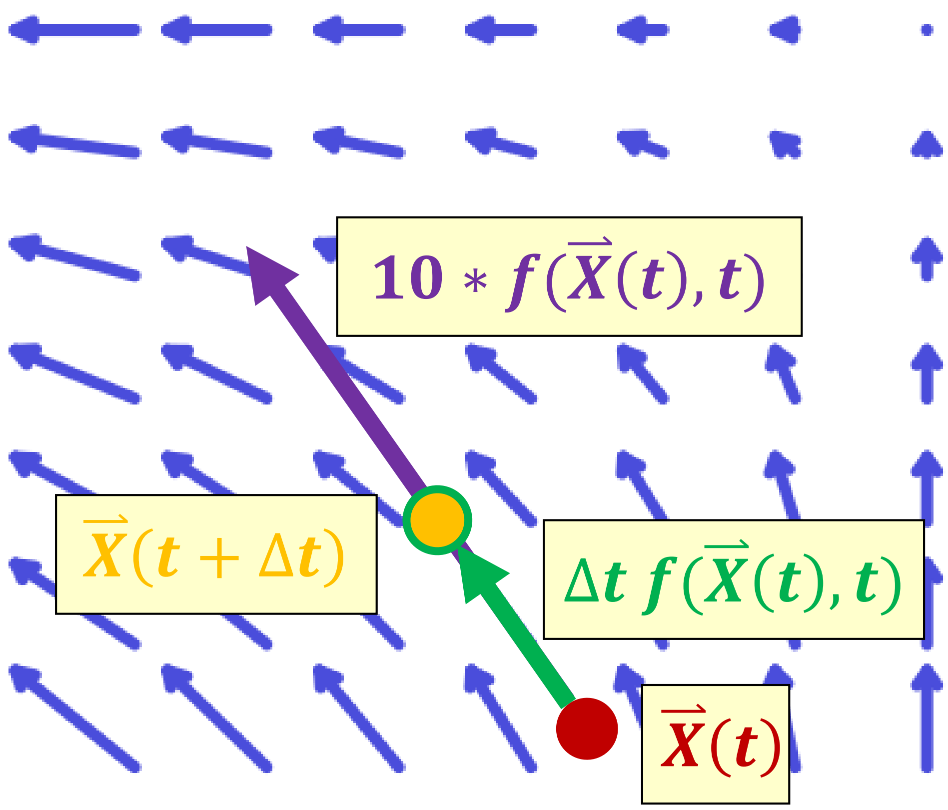
Given that $\vec{X}_0 = \vec{X}(t_0)$

Compute $\vec{X}(t)$ **for** $\mathrm{t} > \mathrm{t}_0$

$$\Delta t = \mathrm{t}_i - \mathrm{t}_{i-1}$$

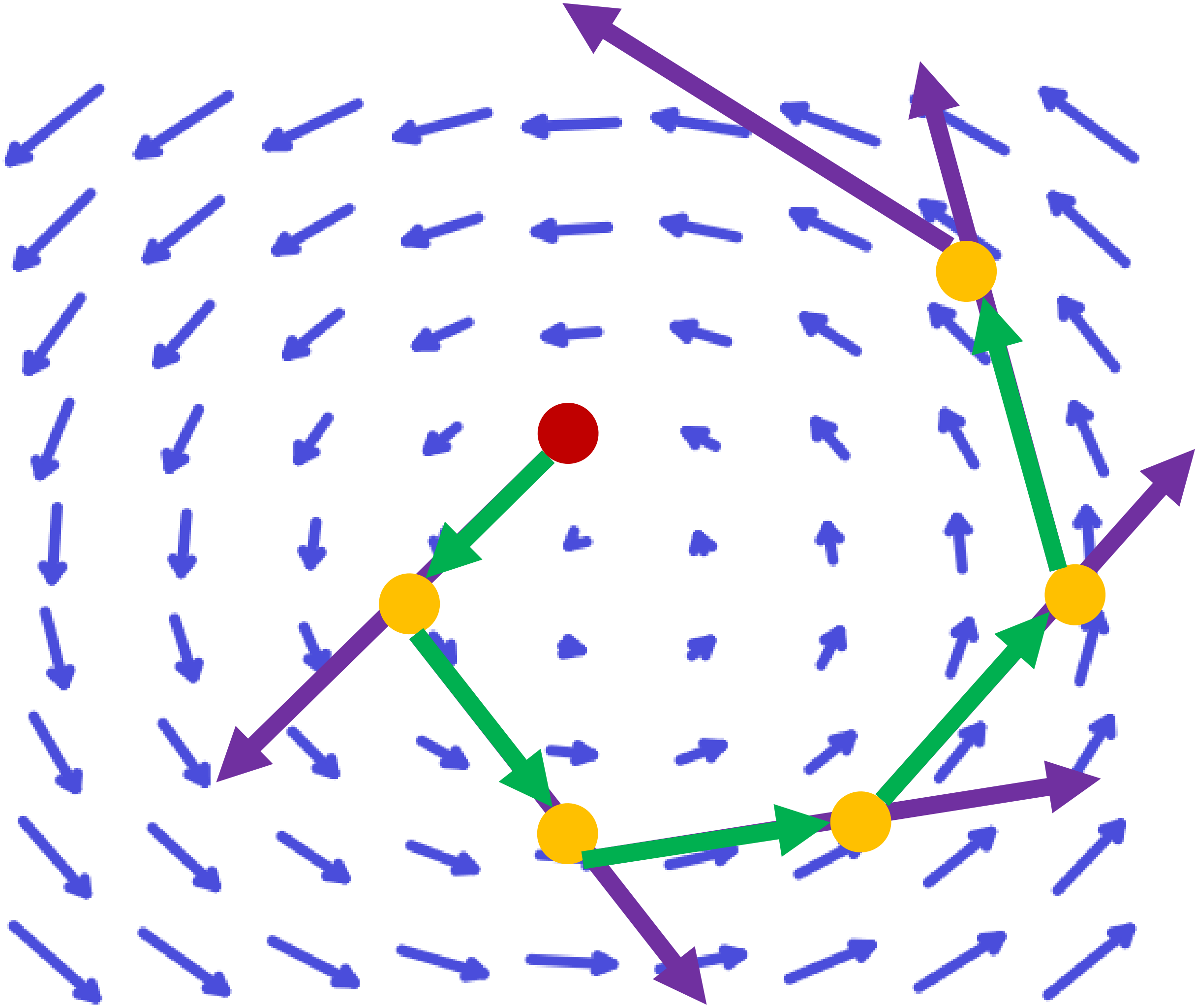
$$\Delta\vec{X}(\mathrm{t}_{i-1}) = \Delta t\, f(\vec{X}(\mathrm{t}_{i-1}), \mathrm{t}_{i-1})$$

$$\vec{X}_i = \vec{X}_{i-1} + \Delta t\, f(\vec{X}_{i-1}, \mathrm{t}_{i-1})$$



Explicit Euler Problems

- Solution spirals out
  - *Even with small time steps*
  - *Although smaller time steps are still better*

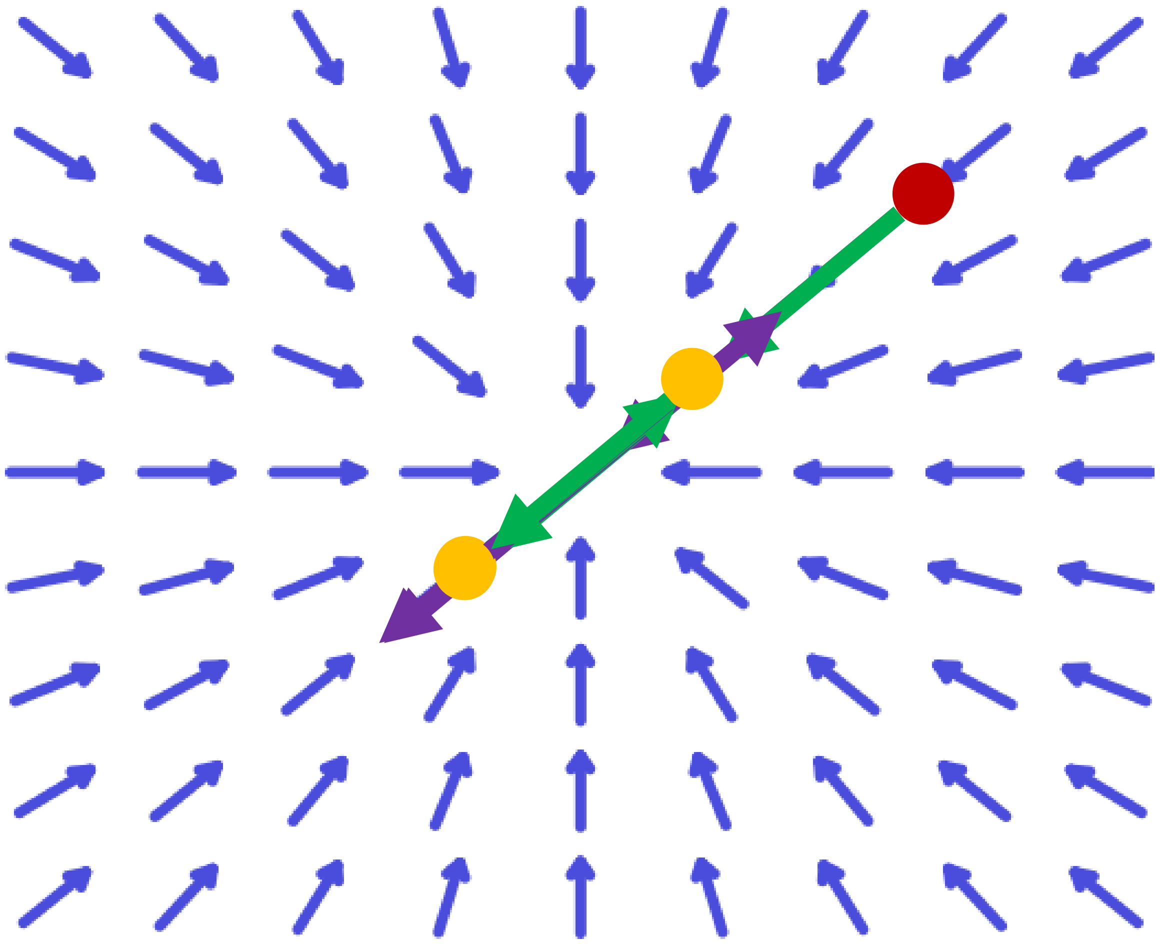


Definition: Explicit

- *Closed-form/analytic solution*
- no iterative solve required

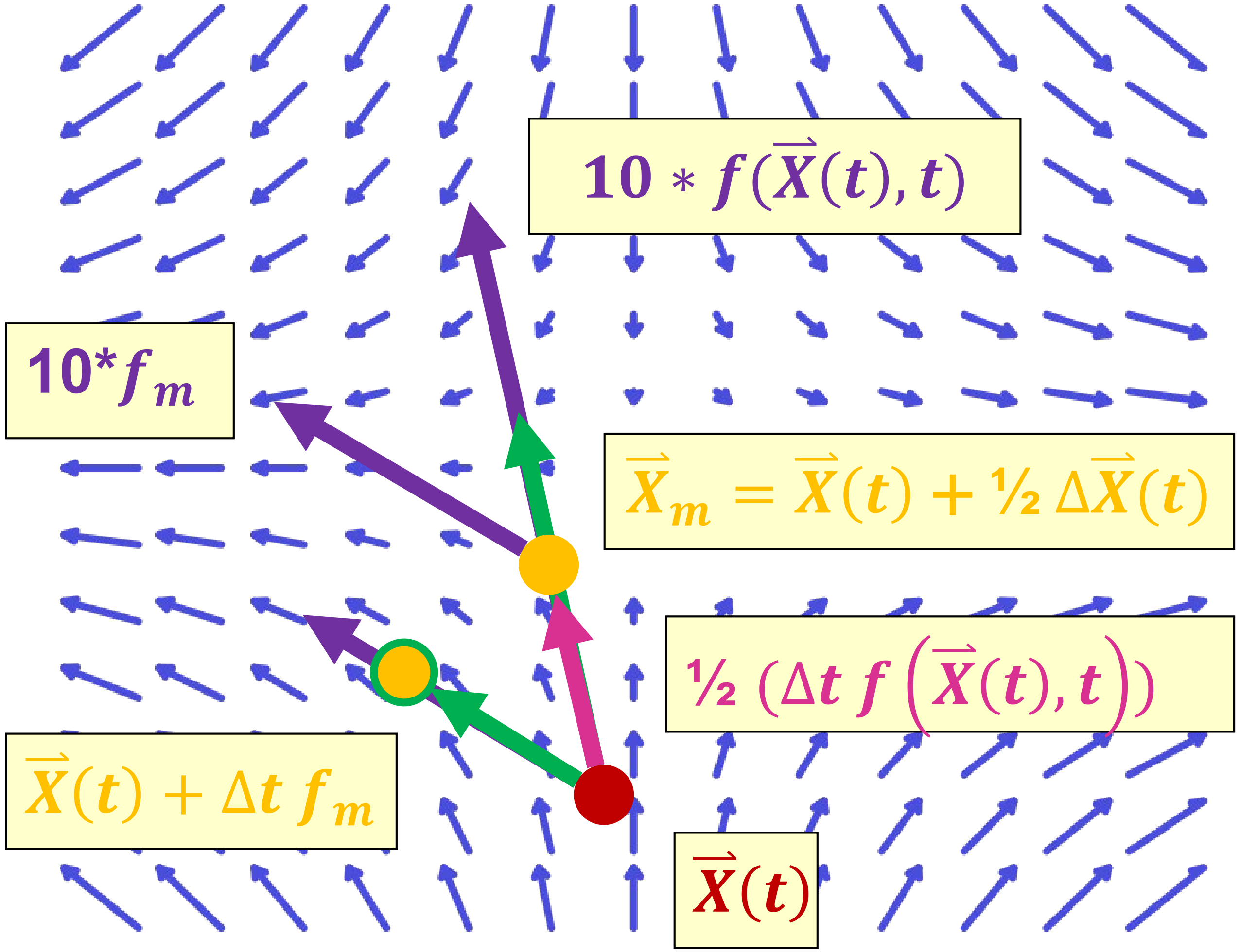
Explicit Euler Problems

- Can lead to **instabilities**



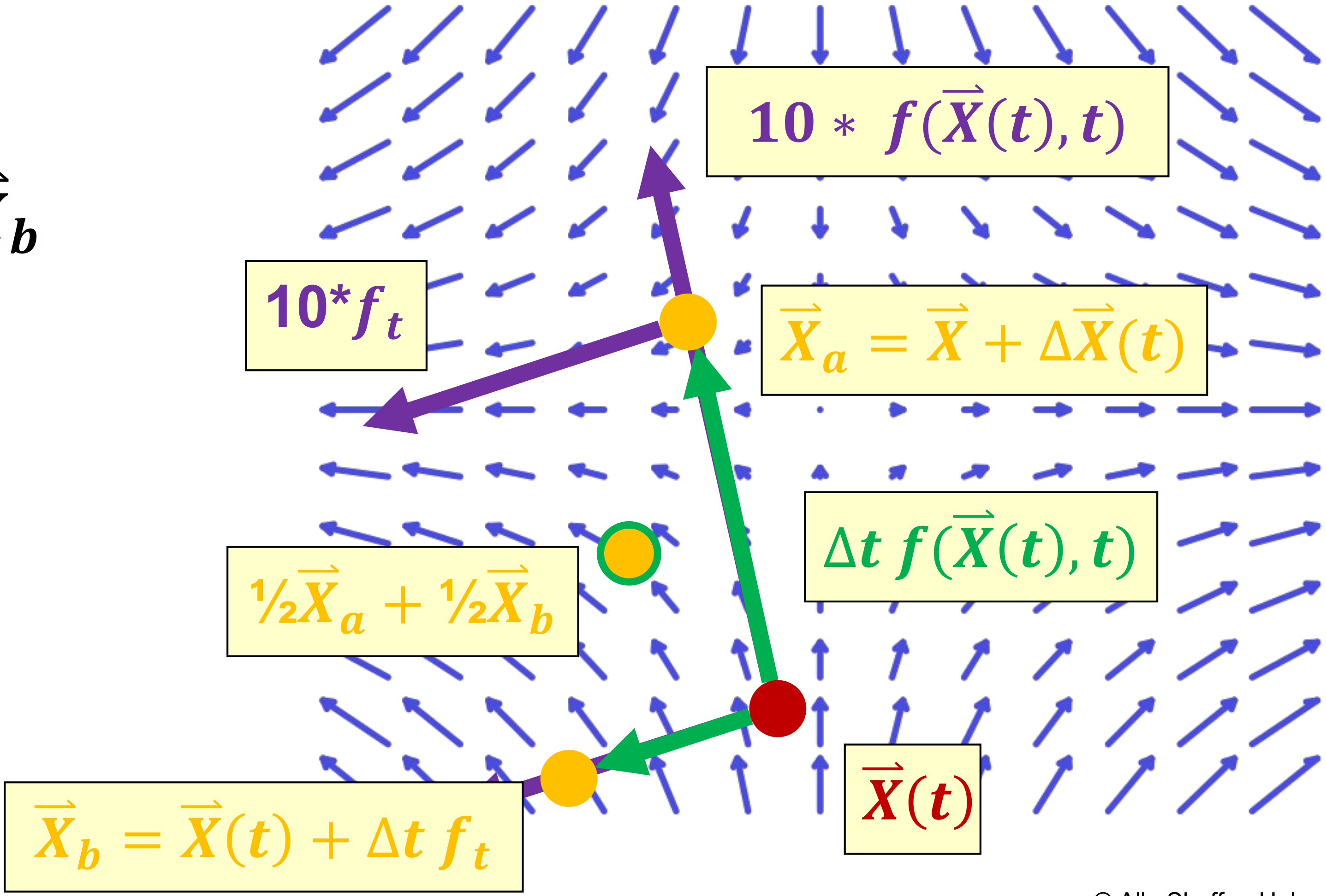
Midpoint Method

1. ½ Euler step
2. evaluate f_m at $\vec{X}_m$
3. full step using f_m



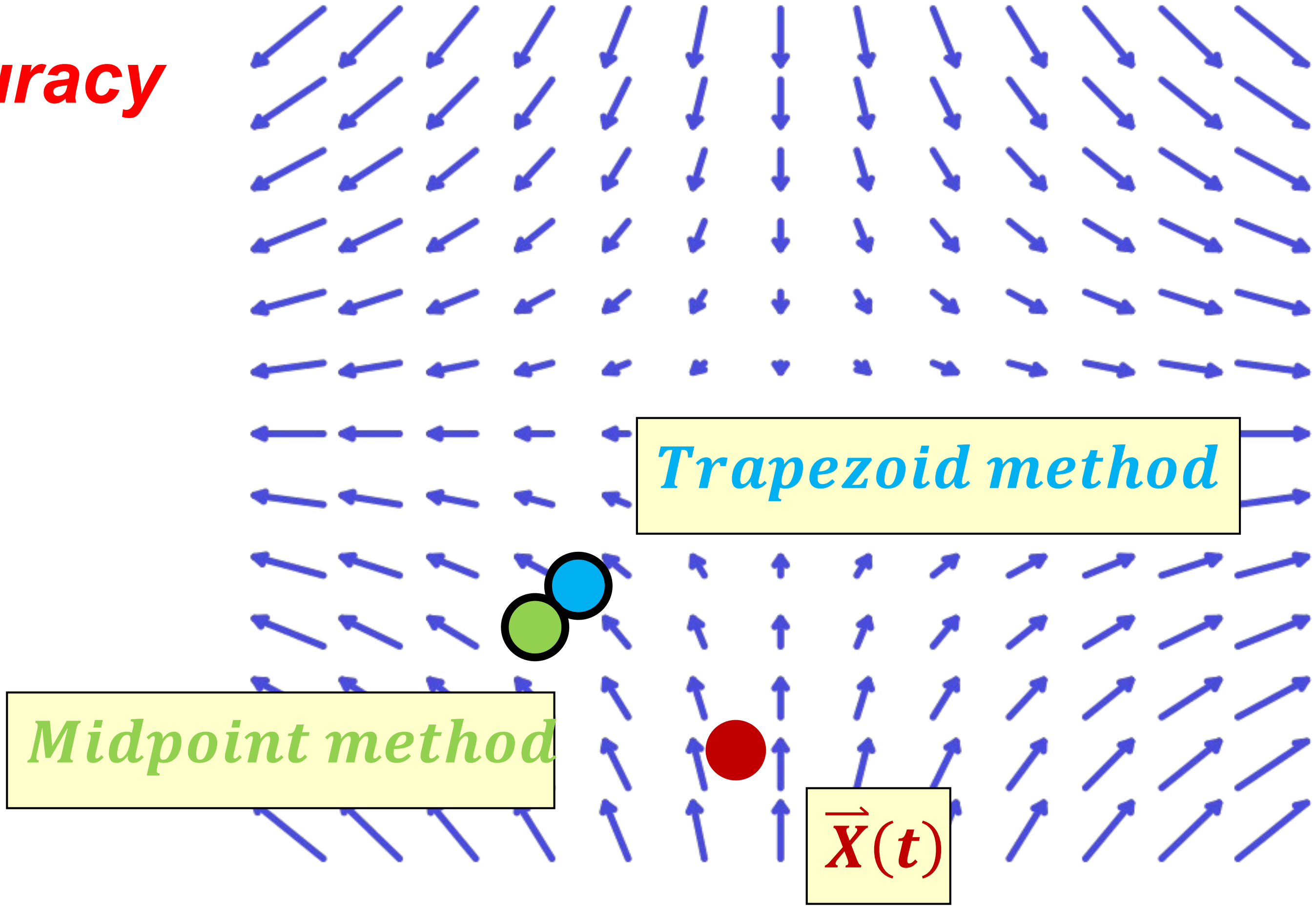
Trapezoid Method

1. full Euler step get $\vec{X}_a$
2. evaluate f_t at $\vec{X}_a$
3. full step using f_t get $\vec{X}_b$
4. average $\vec{X}_a$ and $\vec{X}_b$

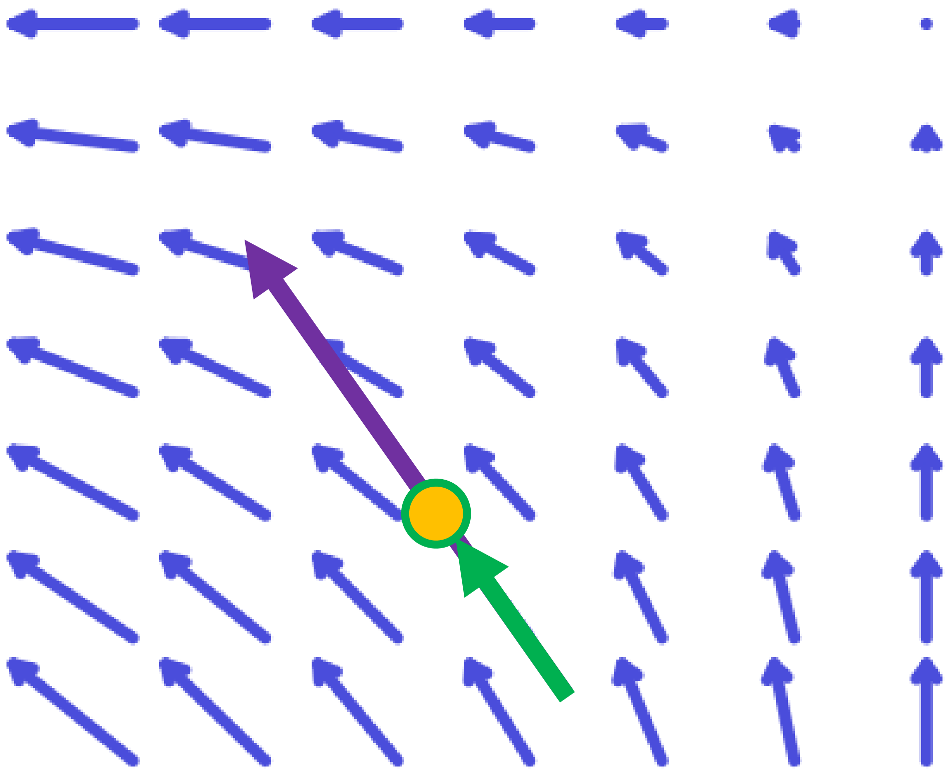


Midpoint & Trapezoid Method

- **Not exactly the same**
  - *But same order of accuracy*

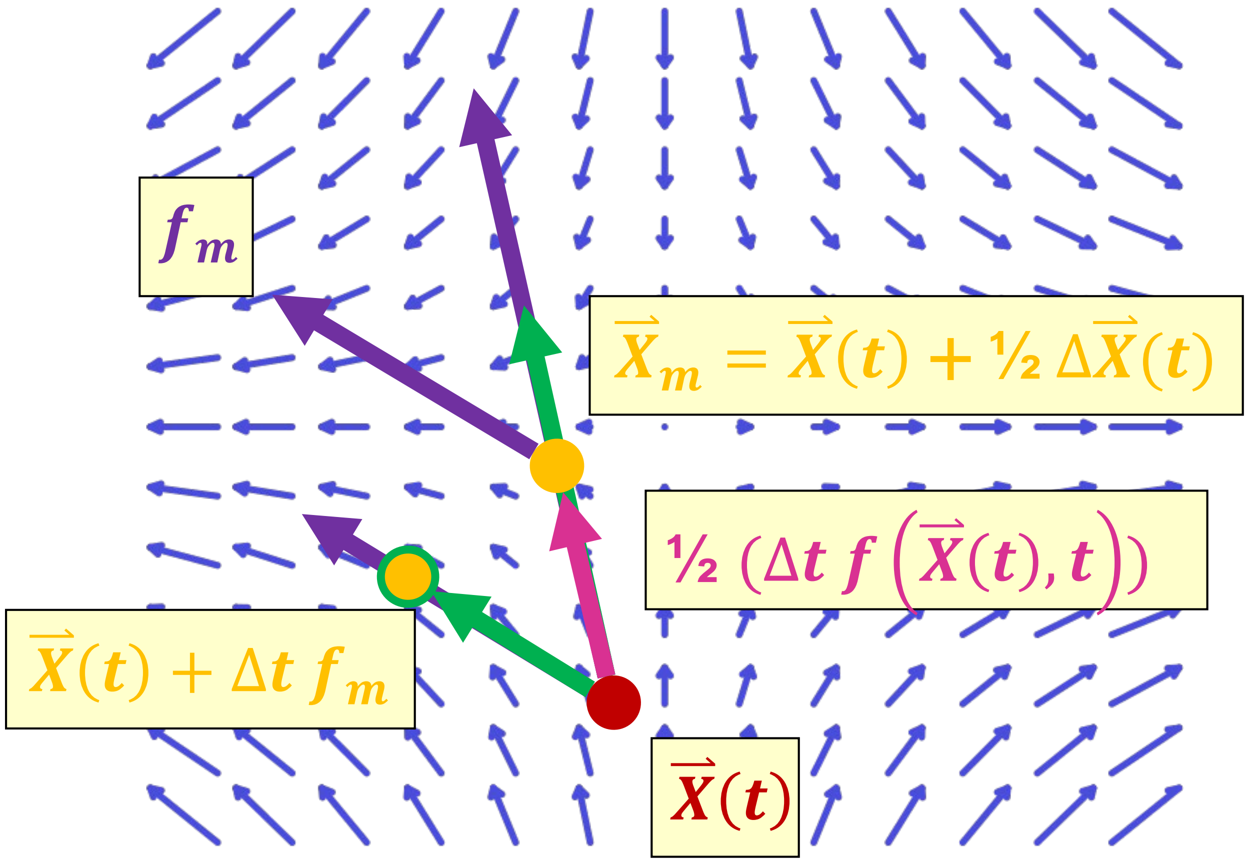


Explicit Euler: Code



```
void takeStep(ParticleSystem* ps, float h)
{
        velocities = ps->getStateVelocities()
        positions = ps->getStatePositions()
        forces = ps->getForces(positions, velocities)
        masses = ps->getMasses()
        accelerations = forces / masses
        newPositions = positions + h*velocities
        newVelocities = velocities  + h*accelerations
        ps->setStatePositions(newPositions)
        ps->setStateVelocities(newVelocities)
}
```

Midpoint Method: Code




```
void takeStep(ParticleSystem* ps, float h)
{
        velocities = ps->getStateVelocities()
        positions = ps->getStatePositions()
        forces = ps->getForces(positions, velocities)
        masses = ps->getMasses()
        accelerations = forces / masses
        midPositions = positions + 0.5*h*velocities
        midVelocities = velocities  + 0.5*h*accelerations
        midForces = ps->getForces(midPositions, midVelocities)
        midAccelerations = midForces / masses
        newPositions = positions + h*midVelocities
        newVelocities = velocities  + h*midAccelerations
        ps->setStatePositions(newPositions)
        ps->setStateVelocities(newVelocities)
}
```

How do we combine force, position & velocity?

$$\vec{X}_0 = (\vec{p}_0,\ \vec{v}_0)$$

$$\frac{\partial}{\partial t}\vec{p}(t) = \vec{v}(t)$$

$$\frac{\partial}{\partial t}\vec{v}(t) = F(\vec{p}(t), \vec{v}(t), t)$$

Requires two: velocity field & force field

$$\frac{\partial}{\partial t}\vec{X}(t) = f(\vec{X}(t), t)$$

Given that $\vec{X}_0 = \vec{X}(t_0)$

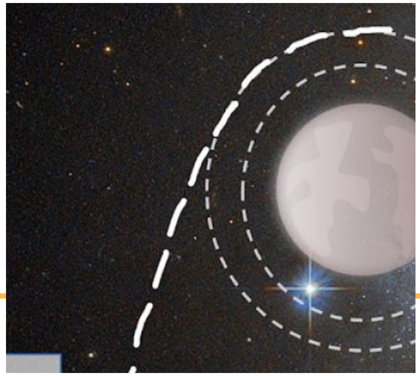
Compute $\vec{X}(t)$ **for** $t > t_0$

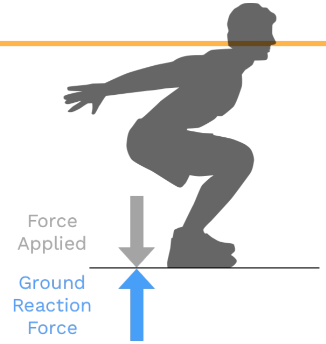
e.g., $\Delta\vec{X}(t) = f(\vec{X}(t), t)\Delta t$

Group work: Draw these two fields for an example setting

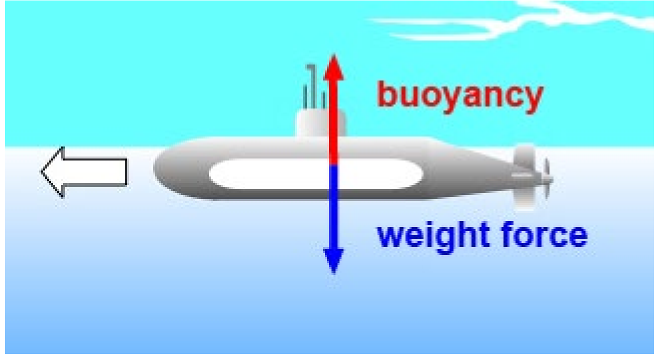
- groups of 3-4, pen & paper, 5-10 minutes + presentation

Possible scenarios



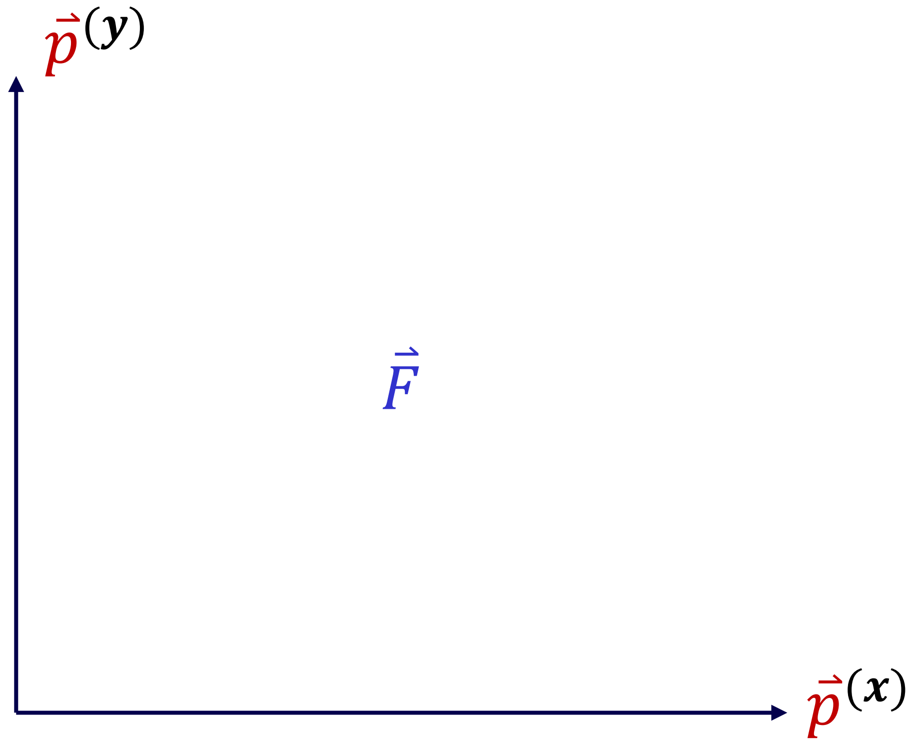


- **Gravity in space**
  only force: gravity towards the center; no resistance
- **Gravity on a flat earth**
  e.g. jump and run, how to model the ground?
- **A train decelerating**
  initial velocity, viscous damping; no gravity
- **A submarine**
  buoyancy only; no resistance
- **Spongebob collision with ground**
  (spring & dampers; no gravity)

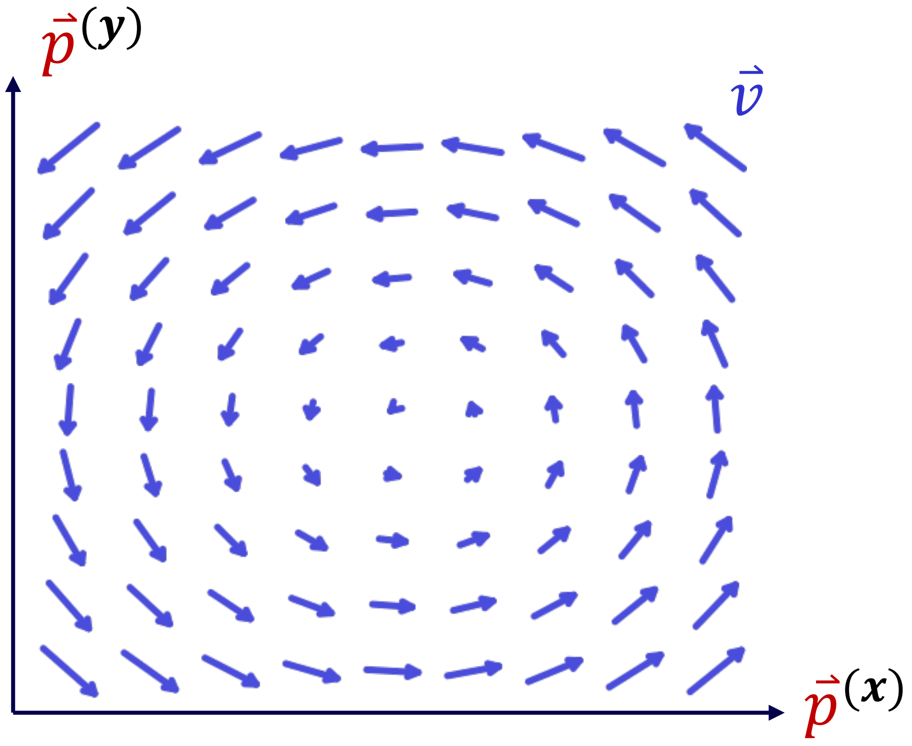


Possible scenarios (difficult!)

- **Balloon in strong wind**
  negligible momentum, how to derive the force?

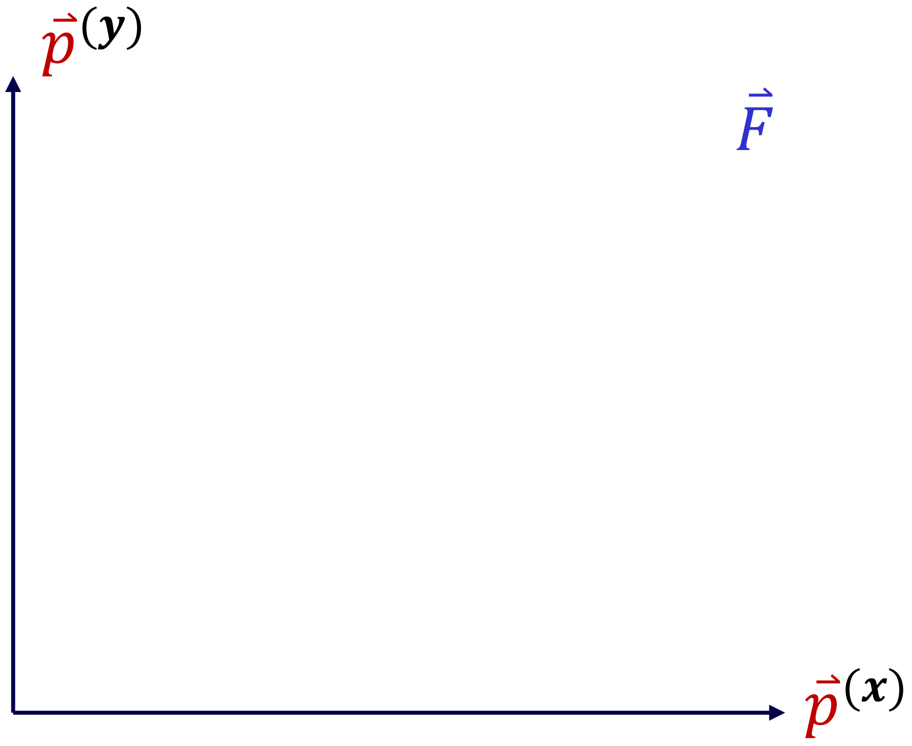


Force field?

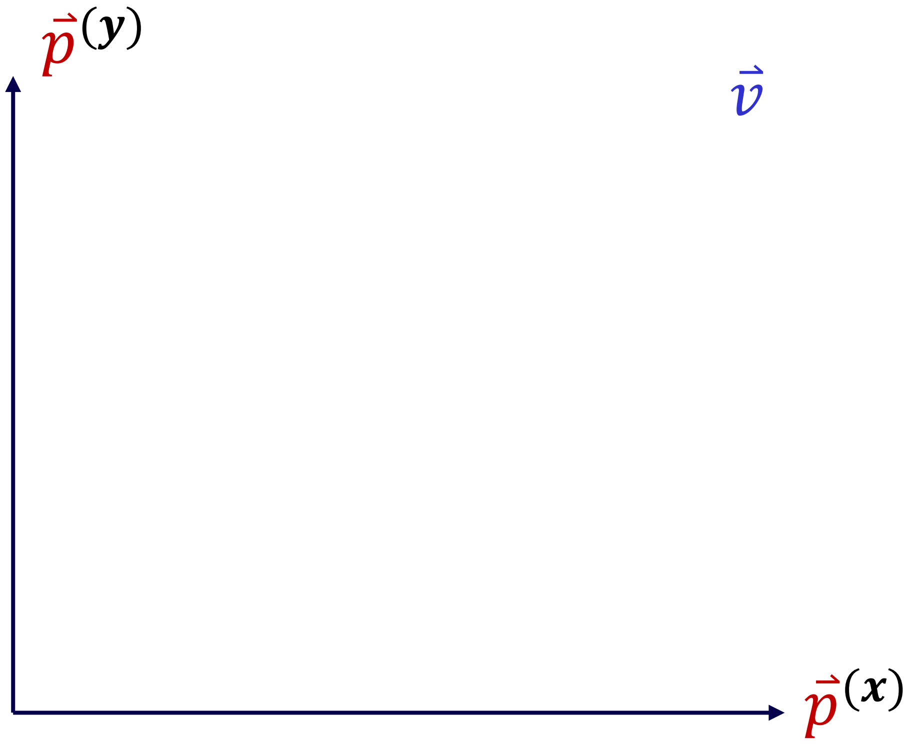


Velocity field

Your scenario - template



Force field



Velocity field

Recap: Forward Euler

Forces only $\vec{F} = ma$

$$d_t = t_{i+1} - t_i$$

$$\vec{v}_{i+1} = \vec{v}_i + (\vec{F}(t_i)/m)d_t$$

$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}_i d_t$$

acceleration $= \frac{\partial v}{\partial t}$

get values at time t_{i+1} from values at time t_i

Issues? Alternatives?

Idea: Backwards Euler

- **Viscous damping**

  $F = -bv$

- **Spring & dampers**

  $F = -kx - bv$

$$d_t = t_{i+1} - t_i$$
$$\vec{v}_{i+1} = \vec{v}_i + (\vec{F}(t_{i+1})/m)d_t$$
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}_{i+1}d_t$$

get values at time t_{i+1} from states at time t_i and forces at t_{i+1}

Issues?

Implicit (Backward) Euler:

- Use forces at destination

Solve system of equations

$$\frac{\partial}{\partial t}\begin{bmatrix}\vec{x}\\ \vec{v}\end{bmatrix} = \begin{bmatrix}\vec{v}\\ \Sigma\vec{F}/m\end{bmatrix}$$

$$\boxed{\begin{aligned} \boldsymbol{x_{n+1}} &= \boldsymbol{x_n} + \boldsymbol{h}\, \boldsymbol{v_{n+1}} \\ \boldsymbol{v_{n+1}} &= \boldsymbol{v_n} + \boldsymbol{h}\left(\frac{\boldsymbol{F_{n+1}}}{\boldsymbol{m}}\right) \end{aligned}}$$

- Types of forces:
  - ***Gravity***

    $$F = \begin{bmatrix}0\\ -mg\end{bmatrix}$$

  - ***Viscous damping***

    $$F = -bv$$

  - ***Spring & dampers***

    $$F = -kx - bv$$

Implicit (Backward) Euler:

- Use forces at destination + **velocity at the destination**

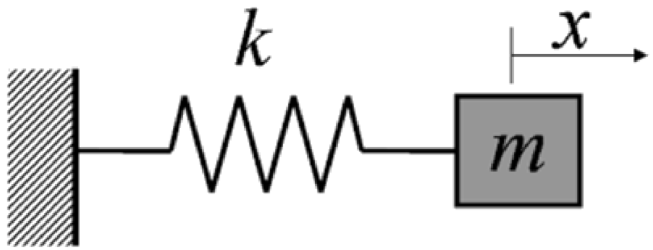
Solve system of equations

$$\frac{\partial}{\partial t}\begin{bmatrix}\vec{x}\\ \vec{v}\end{bmatrix} = \begin{bmatrix}\vec{v}\\ \Sigma\vec{F}/m\end{bmatrix}$$

$$\boxed{\begin{aligned} x_{n+1} &= x_n + h\, v_{n+1} \\ v_{n+1} &= v_n + h\left(\frac{F_{n+1}}{m}\right)\end{aligned}}$$

Example: Spring Force

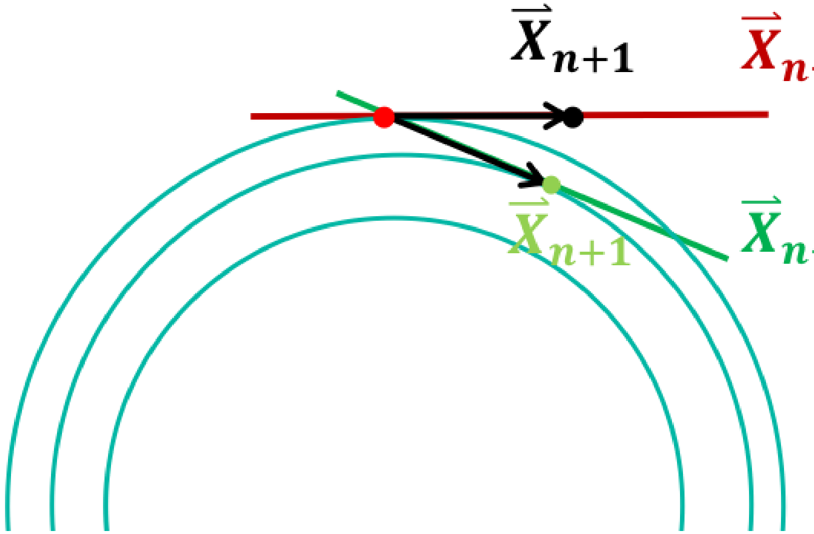
$F = -kx$



$$\boxed{\begin{aligned} x_{n+1} &= x_n + h\, v_{n+1} \\ v_{n+1} &= v_n + h\left(\frac{-k\, x_{n+1}}{m}\right)\end{aligned}}$$

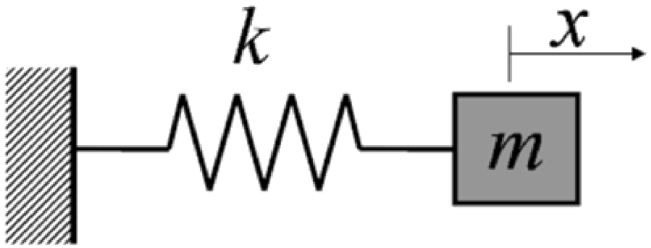
Analytic or iterative solve?

Forward vs Backward



$$\vec{X}_{n+1} = \vec{X}_n + \Delta t\, f(\vec{X}_n)$$

$$\vec{X}_{n+1} = \vec{X}_n + \Delta t\, f(\vec{X}_{n+1})$$



Forward Euler

$$\boxed{\begin{aligned} x_{n+1} &= x_n + h\, v_n \\ v_{n+1} &= v_n + h\left(\frac{-k\, x_n}{m}\right) \end{aligned}}$$

Backward Euler

$$\boxed{\begin{aligned} x_{n+1} &= x_n + h\, v_{n+1} \\ v_{n+1} &= v_n + h\left(\frac{-k\, x_{n+1}}{m}\right) \end{aligned}}$$

Could one apply the Trapezoid Method?

Particles: Newtonian Physics as First-Order DE

- **Motion of many particles?**

$$\frac{\partial}{\partial t}\begin{bmatrix}\overrightarrow{x_1}\\ \overrightarrow{v_1}\\ \overrightarrow{x_2}\\ \overrightarrow{v_2}\\ \vdots\\ \overrightarrow{x_n}\\ \overrightarrow{v_n}\end{bmatrix} = \begin{bmatrix}\overrightarrow{v_1}\\ \overrightarrow{F_1}/m_1\\ \overrightarrow{v_2}\\ \overrightarrow{F_2}/m_2\\ \vdots\\ \overrightarrow{v_n}\\ \overrightarrow{F_n}/m_n\end{bmatrix}$$

- **Interaction of particles?**

Multiple-particle collision

- ***naïve implementation is likely unstable***
    - *Objects pushing inside each other*

- ***Further reading:***
- https://box2d.org/publications/
    - *In particular https://box2d.org/files/ErinCatto_ModelingAndSolvingConstraints_GDC2009.pdf*

Simulation Basics

Simulation loop…

1. *Equations of Motion*
2. *Numerical integration*
3. *Collision detection*
4. *Collision resolution*

Collisions

- Collision detection
  - *Broad phase: AABBs, bounding spheres*
  - *Narrow phase: detailed checks*
- Collision response
  - *Collision impulses*
  - *Constraint forces: resting, sliding, hinges, ….*

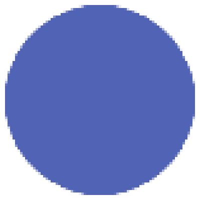
Basic Particle Simulation (first try)

Forces only $\vec{F} = ma$

$$d_t = t_{i+1} - t_i$$

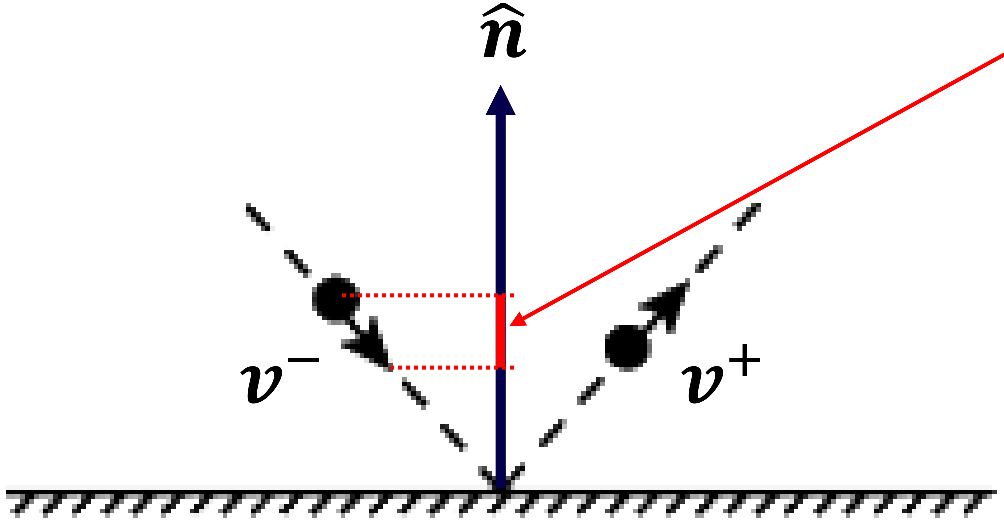
$$\vec{v}_{i+1} = \vec{v}(t_i) + (\vec{F}(t_i)/m)d_t$$

$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}(t_{i+1})d_t$$



Recap: Particle-Plane Collisions (in terms of vel.)

- ***Change in direction of normal***



Velocity along normal (v projected on normal by the dot product)

Frictionless

$$\Delta v = 2\underbrace{(v^- \cdot \hat{n})}_{\text{Velocity along normal}}\hat{n}$$

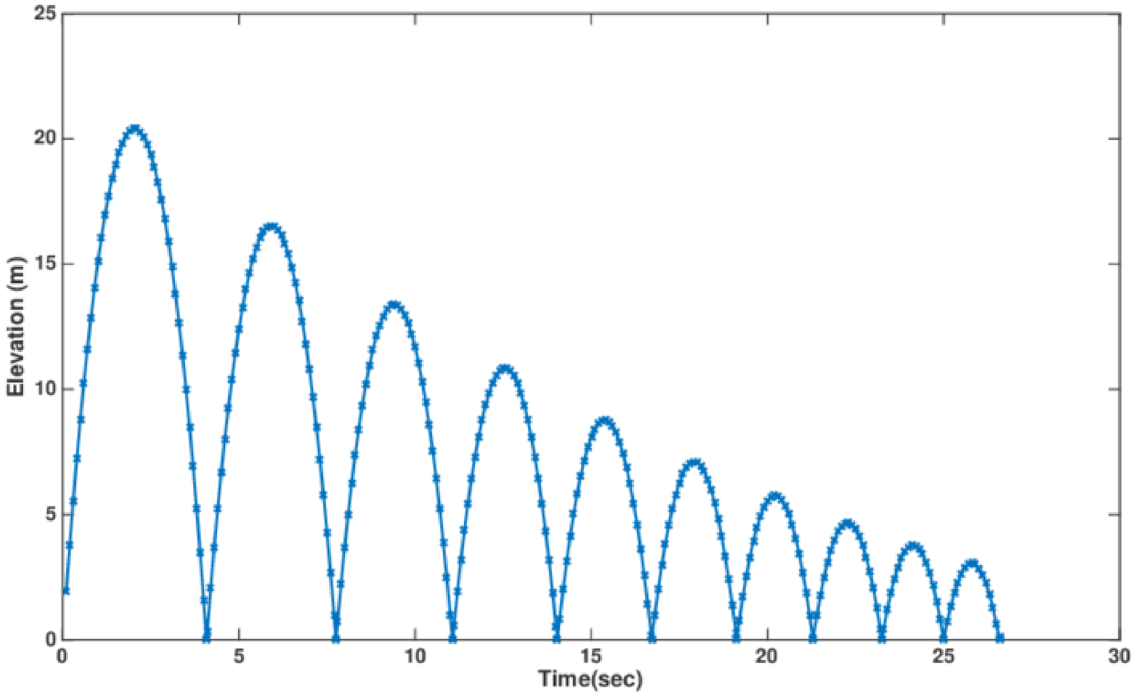
Apply change along normal (magnitude times direction)

$$v^+ = v^- + \Delta v$$

Why use 'Impulse'?

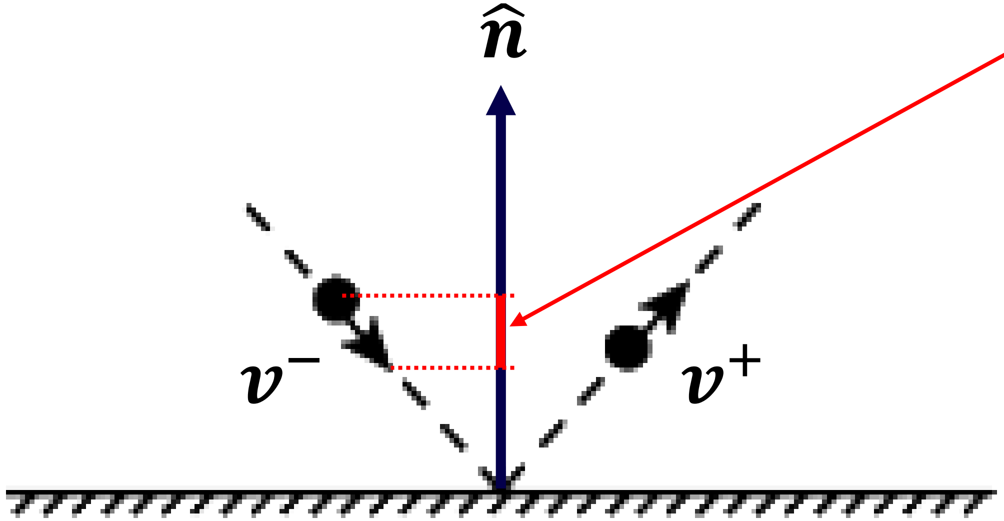
- ***Integrates with the physics solver***

- ***How to integrate damping?***



Recap: Particle-Plane Collisions (in terms of vel.)

- ***Change in direction of normal***



Velocity along normal
(v projected on normal
by the dot product)

Frictionless

Apply change
along normal
(magnitude
times direction)

$$\Delta v = 2(v^- \cdot \hat{n})\hat{n}$$

$$v^+ = v^- + \Delta v$$

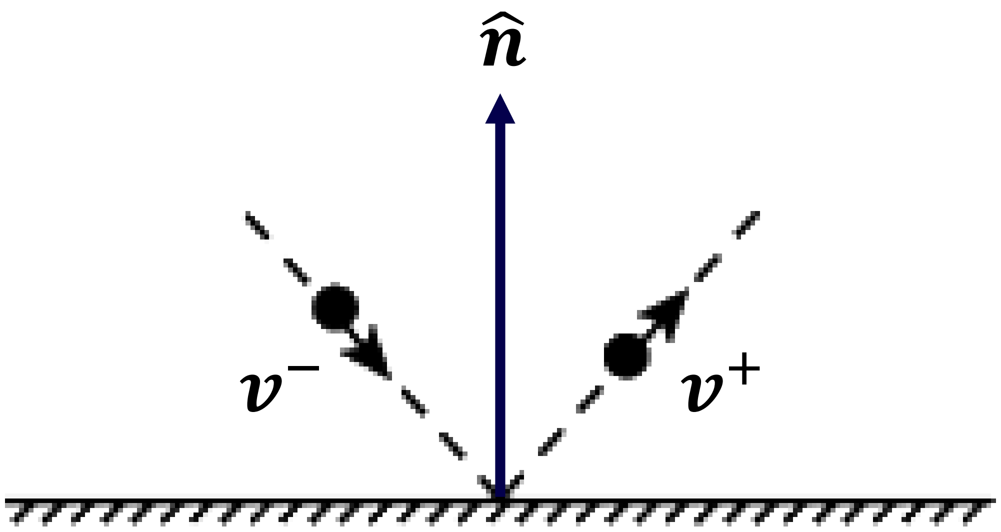
Loss of energy

$$\Delta v = (1+\epsilon)(v^- \cdot \hat{n})\hat{n}$$

Particle-Plane Collisions

- ***Apply an*** ***'impulse'*** ***of magnitude j***
  - Inversely proportional to mass of particle
- ***In direction of normal***

Impulse in physics: Integral of F over time
In games: an instantaneous step change (not physically possible), i.e., the force applied over one time step of the simulation



$$j = (1+\epsilon)(v^- \cdot \hat{n})m$$

$$\vec{j} = j\,\hat{n}$$

$$v^+ = \frac{\vec{j}}{m} + v^-$$

What is the effect of ϵ ?

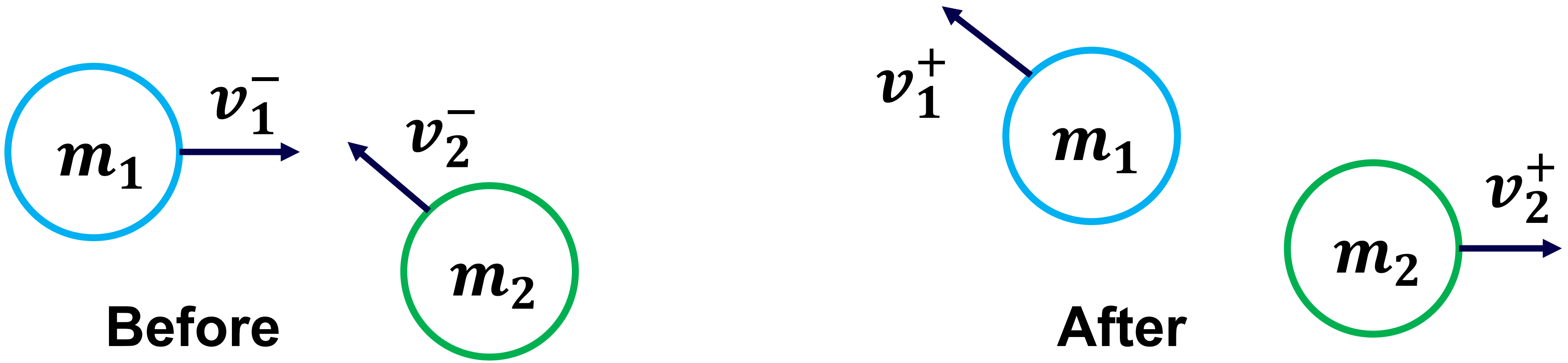
Hint for A2: Check for velocity direction!

- *Only collide if objects are moving towards each other*
- *Ignore collision if moving away from each other*

Q: Should I work in terms of velocity or forces?

Particle-Particle Collisions (radius=0)

- Particle-particle **frictionless elastic impulse response**



- Momentum is **preserved**

$$m_1 v_1^- + m_2 v_2^- = m_1 v_1^+ + m_2 v_2^+$$

- Kinetic energy is **preserved**

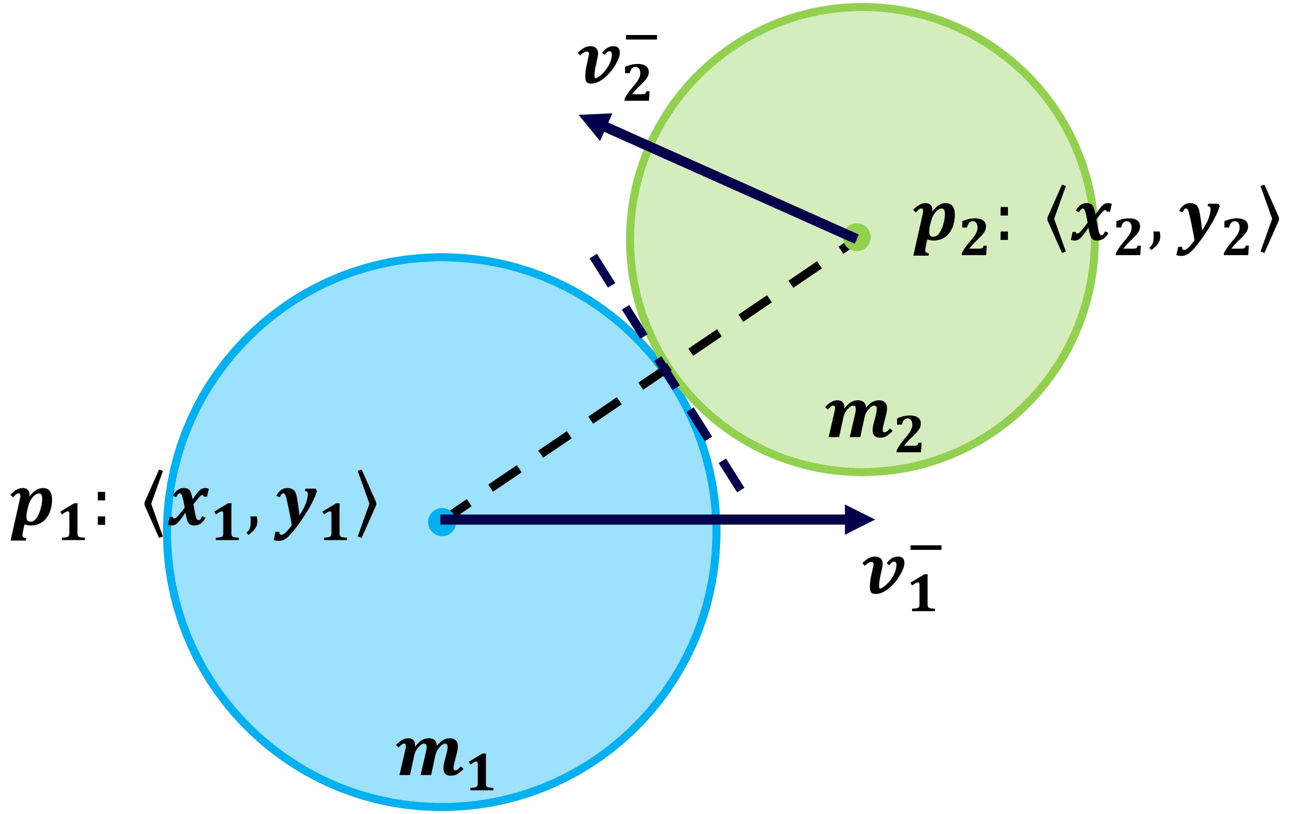
$$½\, m_1 {v_1^-}^2 + ½\, m_2 {v_2^-}^2 = ½\, m_1 {v_1^+}^2 + ½\, m_2 {v_2^+}^2$$

- Velocity is **preserved in tangential direction**

$$t \circ v_1^- = t \circ v_1^+ , \qquad t \circ v_2^- = \mathbf{t} \circ v_2^+$$

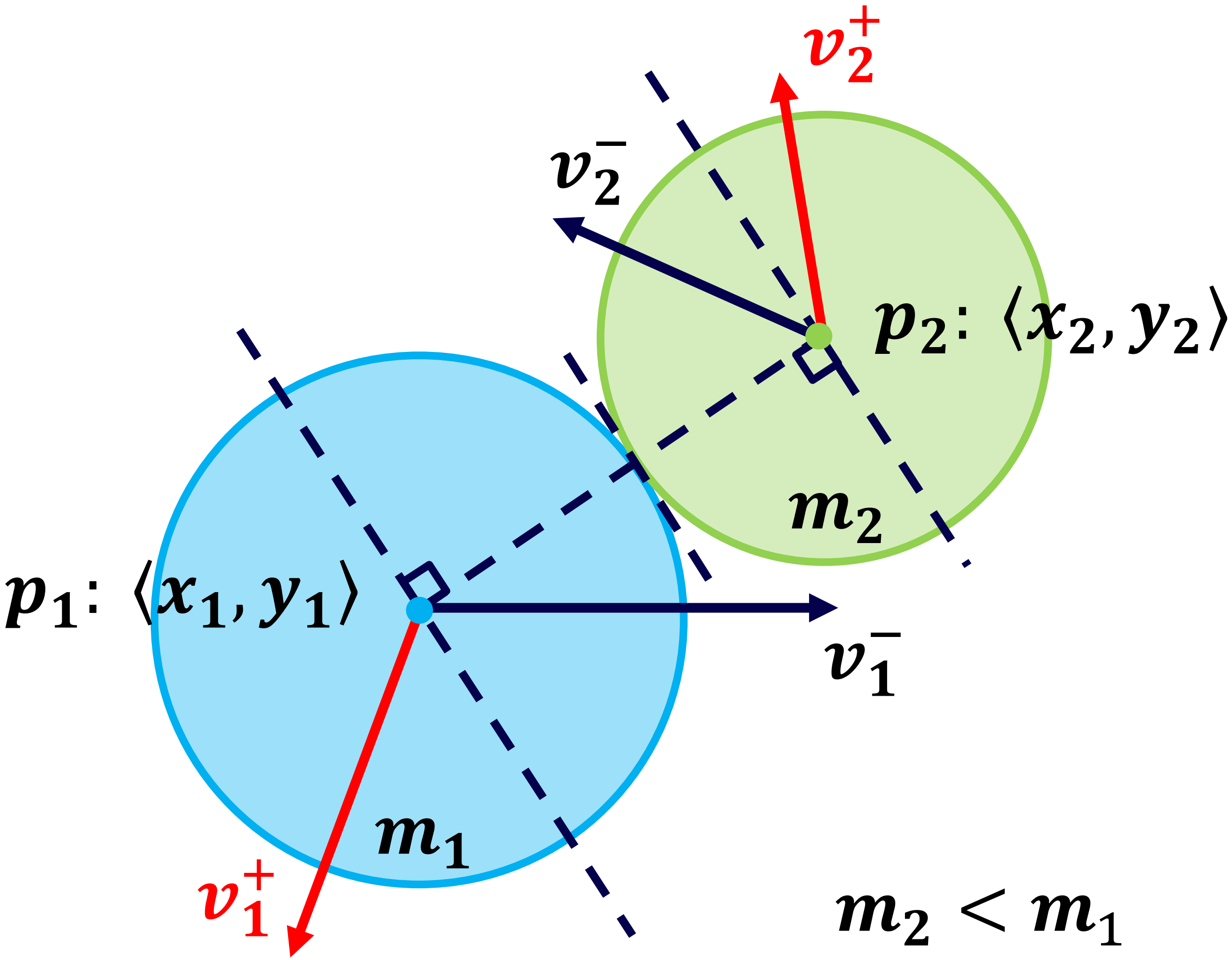
Particle-Particle Collisions (radius >0)

- What we know…
  - *Particle centers*
  - *Initial velocities*
  - *Particle Masses*
- What we can calculate…
  - *Contact normal*
  - *Contact tangent*



Particle-Particle Collisions (radius >0)

- Impulse **direction** reflected across **tangent**
- Impulse **magnitude** proportional to **mass of other particle**



$m_2 < m_1$

Particle-Particle Collisions (radius >0)

- **More formally…**

$$\boldsymbol{v}_1^+ = \boldsymbol{v}_1^- - \frac{2m_2}{m_1+m_2}\frac{\langle \boldsymbol{v}_1^- - \boldsymbol{v}_2^- \rangle \cdot \langle \boldsymbol{p}_1 - \boldsymbol{p}_2 \rangle}{\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|^2}\langle \boldsymbol{p}_1 - \boldsymbol{p}_2 \rangle$$

$$\boldsymbol{v}_2^+ = \boldsymbol{v}_2^- - \frac{2m_1}{m_1+m_2}\frac{\langle \boldsymbol{v}_2^- - \boldsymbol{v}_1^- \rangle \cdot \langle \boldsymbol{p}_2 - \boldsymbol{p}_1 \rangle}{\|\boldsymbol{p}_2 - \boldsymbol{p}_1\|^2}\langle \boldsymbol{p}_2 - \boldsymbol{p}_1 \rangle$$

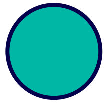
- This is in terms of velocity, what would the corresponding impulse be?

Hints: Sequential impulse updates

- ***Important to update velocity right after computing constraint/forces***
- ***Important to update the velocity of both objects at the same time for a collision event***
- ***Clamping does not work with bouncing constraints***
- ***Accumulated clamping has no effect in the simple stacking test***
  - *Requires Bounce/restitution to be implemented as velocity bias (not updated during inner iterations)*
  - *Since the other half of the velocity is taken care of by the contact constraint*
- ***Pseudo velocities don't give a huge improvement***
- Perhaps different when including rotational motion?

Rigid Body Dynamics (rotational motion of objects?)

- **From particles to rigid bodies…**



Particle

$$state = \begin{cases} \vec{x}\; position \\ \vec{v}\; velocity \end{cases}$$

$\mathbb{R}^4$ in 2D
$\mathbb{R}^6$ in 3D

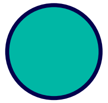
Rigid body

$$state = \begin{cases} \vec{x}\; position \\ \vec{v}\; velocity \\ R\; rotation\; matrix\; 3x3 \\ \vec{w}\; angular\; velocity \end{cases}$$

$\mathbb{R}^{12}$ in 3D

Rigid Body Dynamics

- From particles to rigid bodies…



Angular vel.

Newton's equations of motion

$$\Sigma\vec{F} = m\vec{a}$$

$$\begin{bmatrix} m & & \\ & m & \\ & & m \end{bmatrix}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \Sigma\vec{F} \end{bmatrix}$$

$$M\vec{a} = \Sigma\vec{F}$$

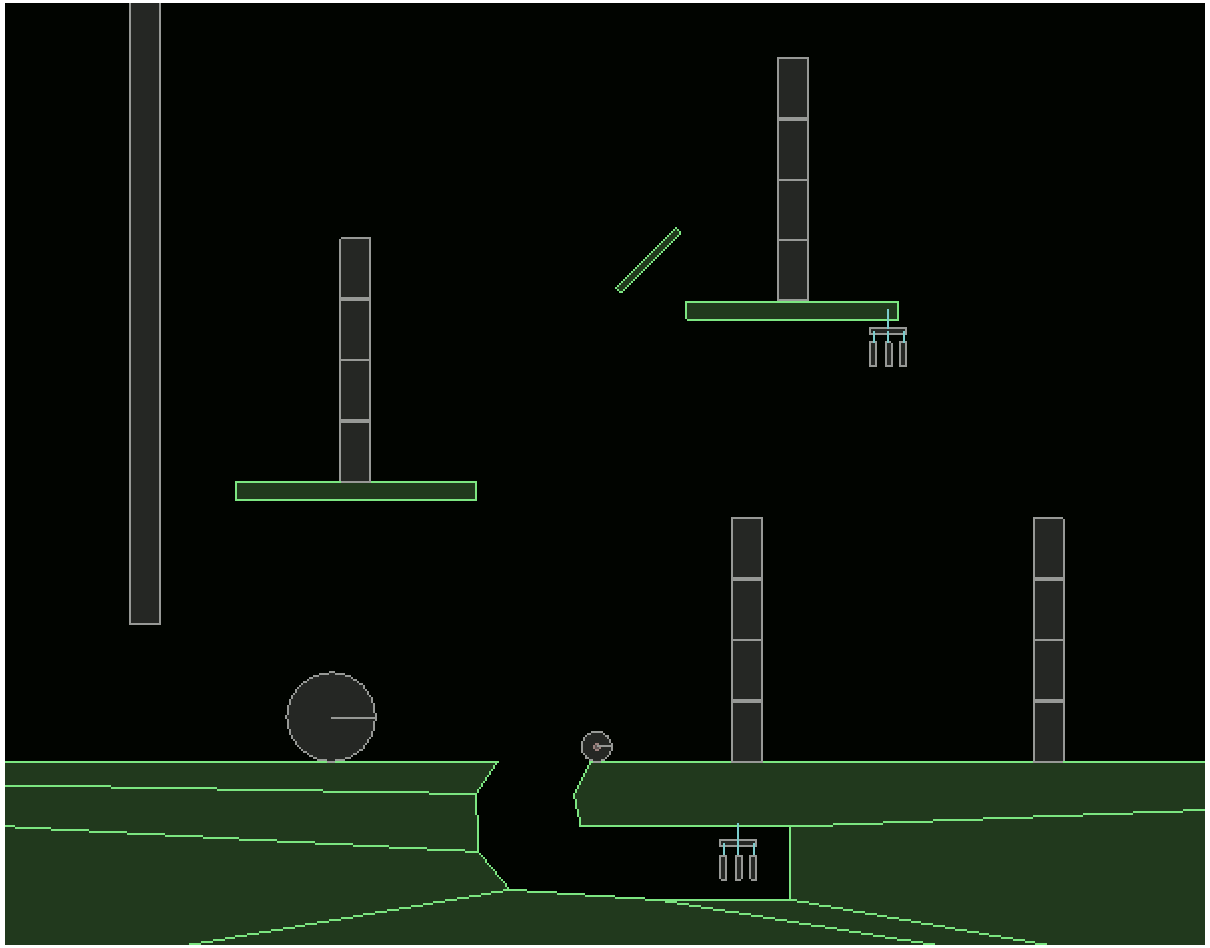
Newton-Euler equations of motion

$$\begin{bmatrix} m & & & \\ & m & & \\ & & m & \\ & & & \mathbf{I} \end{bmatrix}\begin{bmatrix} a_x \\ a_y \\ a_z \\ w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} \Sigma\vec{F} \\ \\ \end{bmatrix}$$

Inertia tensor $\Sigma\vec{\tau} - \vec{w} \times \mathbf{I}\vec{w}$

Box2D – an excellent 2D physics engine

Particle and rigid body dynamics!



Organizational

- ***M1 face2face grading this week***
  - have a laptop ready, with the game running

- ***M1 Team presentations***
  - Add your slides
    https://docs.google.com/presentation/d/1Y4h7ns1uFLIyWO-
    ing
    - 1 minute presentation
    - add slides to this presentation
    - if applicable, share screen on zoom for live demo (have everything ready!)
    - sales pitch style
      - hype us about your progress and features
    - all team members must be present
      (such that we all know who works with whom)
    - at least two students must present (different ones for each milestone)
      - e.g. each explaining one feature

Outlook – Guest lectures

11	Mon	13-Nov	Break		
	Wed	15-Nov	Break		
12	Mon	20-Nov	**Lecture: Balancing games** **Guest lecture by Gavin Young (Behaviour Interactive)**	**Milestone 3**	
	Wed	22-Nov	**Lecture: Team-report M3** **Tutorial: Face2Face grading M3**		
13	Mon	27-Nov	**Guest lecture by Yggy King (Blackbird Interactive) on "ECS in practice"** **Cross-play M3**		
	Wed	29-Nov	**Guest lecture by Cloé Veilleux (Relic Entertainment) "Cutting corners in AI"**		
14	Mon	4-Dec	**Lecture: The history and future of game technology** **Guest Lecture by Alex Denford and Cate Mackenzie (SkyBox) "Empowering Creators"**	**M4 submission**	
	Wed	6-Dec	**Lecture: Team-report M3** **Tutorial: Face2Face grading M4**		
Exam slot TBD			**Cross-play M4** **Awards**		