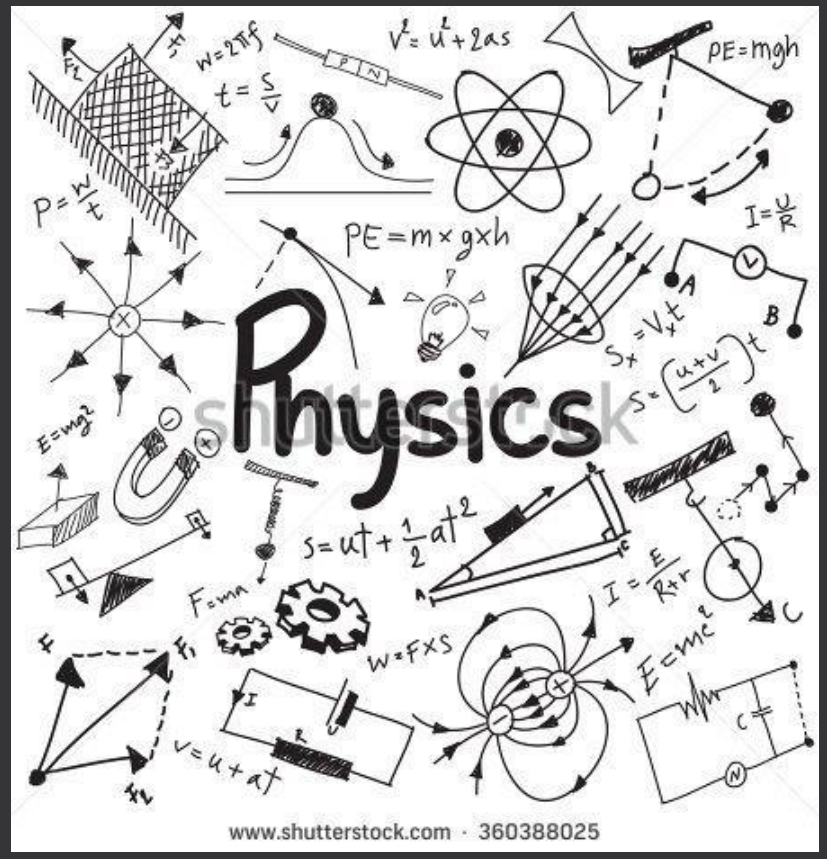
A little physics

Lecture 3

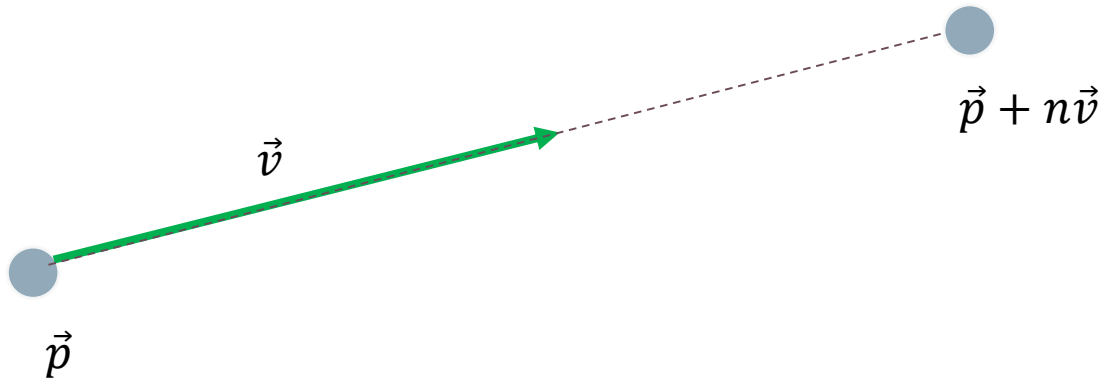


Why physics?

- Every game developer should know it.
- A good chance to *apply* our vector library
- Typically ~50% of students haven't had *any* physics and ~50% haven't had calculus yet.
  - So…physics and calculus lite
  - I'll try to present this in a more applied way (less theory, some "hand-waving")

Velocity

- Speed *and* direction
  - So…a vector!
- Example:
  - $\vec{p}$: the character's position
  - $\vec{v}$: the character's velocity
  - If the character moves with this velocity for **n** seconds…
  - …the character would now be at $\vec{p} + n\vec{v}$.
  - (in the picture, I'm implying that n ~ 2 seconds)
  - Note: this works in *any* dimension!



A word about our game loop

- Two choices:

```
clock = pygame.time.Clock()          # Before game loop
```

  - **Unlimited frame-rate**
    - The game loop runs as fast as possible
    - Depending on scene complexity, the time to perform an **iteration** of the game loop varies.

```
dt = clock.tick() / 1000.0          # Time (in s) since last update
```

    - Advantage: will run on any machine
    - Disadvantage: dt varies (bad for physics [esp. collision handling], animation systems, etc.)
  - **Fixed frame-rate**
    - Each frame, <u>if</u> we're able to reach the target frame rate…
    - …add a small delay

```
dt = clock.tick(60) / 1000.0        # Should be ~ 1/60 second
```

    - Advantage: Predictable
    - Disadvantage: Slow machines will look very laggy

- **n** on the previous slide stands for dt (that we just calculated)

Velocity Integration

- Suppose:
  - we are doing fixed-frame rate dt's
  - a character is moving with velocity v for 5 iterations
- The position at these 5 iterations will be:
  - $\overrightarrow{p1} = \overrightarrow{p0} + \Delta t * \vec{v}$
  - $\overrightarrow{p2} = \overrightarrow{p1} + \Delta t * \vec{v}$
  - $\overrightarrow{p3} = \overrightarrow{p2} + \Delta t * \vec{v}$
  - $\overrightarrow{p4} = \overrightarrow{p3} + \Delta t * \vec{v}$
  - $\overrightarrow{p5} = \overrightarrow{p4} + \Delta t * \vec{v}$
- In code, we'd put this in our game loop:

```
p += dt * v
```

- A mathematician of physicist would say we're doing:

$$\overrightarrow{pi} = \int_{t=0}^{t=i} \vec{v}\, dt$$

Units of Measurement

- The unit of measurement depends on the coordinate system
  - Could be pixels if we're using pygame
  - Could be meters (or feet) if doing a physics simulation
  - In 3d graphics, we usually just say "world units"
- When I say "units", think of the context
- Velocity is often measured in units / s
  - pixels / s in pygame
  - Note how our update formula ($\overrightarrow{p\_new} = \overrightarrow{p\_old} + \overrightarrow{vel} * \Delta t$), the units cancel!

Variable velocity

- Constant velocity = boring!
- When an object changes speed (or direction), it is being subject to **acceleration**
  - No (real-world) object can start / stop instantaneously
  - Instead, it's a *gradual* change.
  - Acceleration is also a vector quantity.
  - Expressed in units / s2
  - To update velocity (with constant acceleration), we use:

$$\overrightarrow{vi} = \int_{t=0}^{t=i} \vec{a}\, dt$$

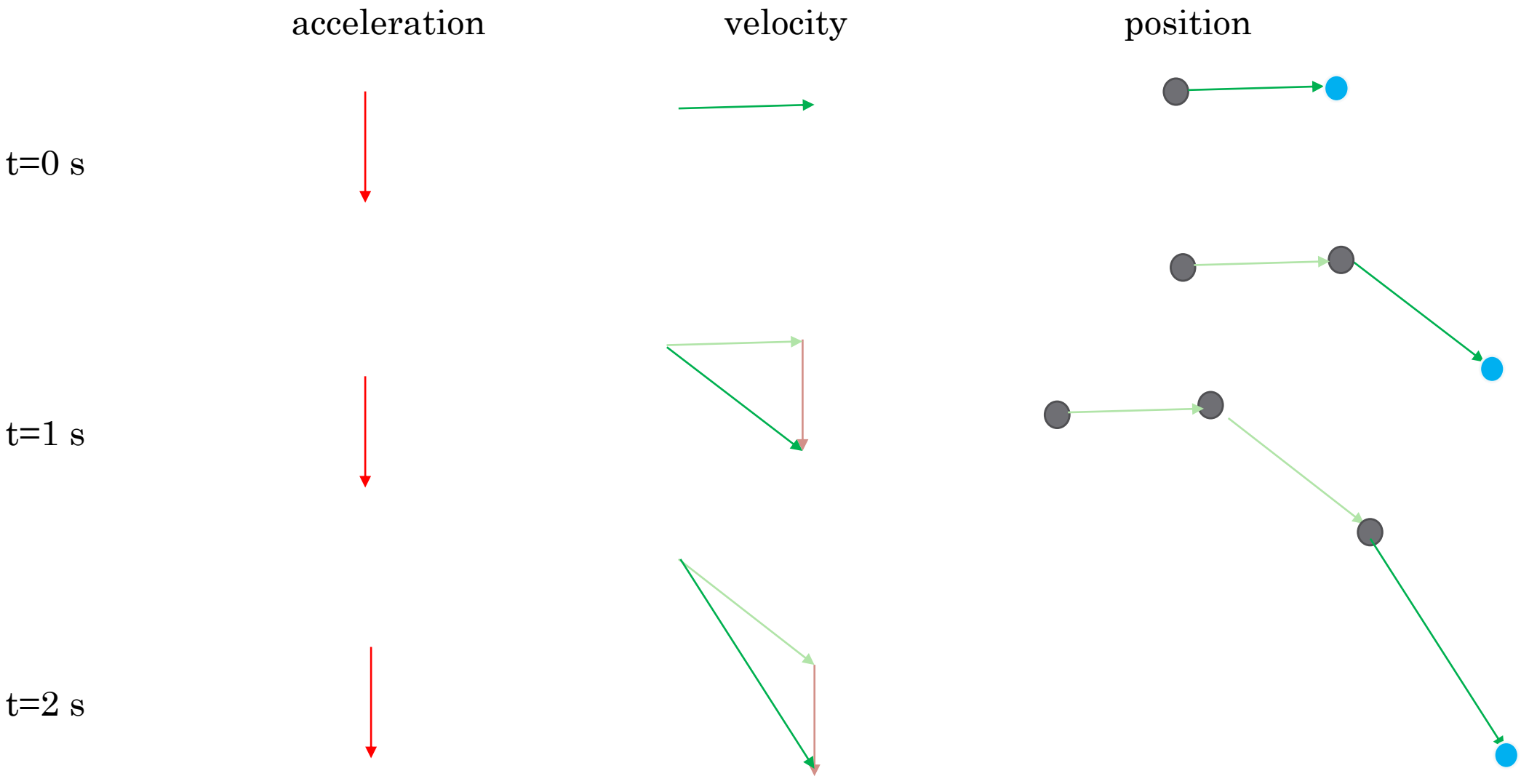
  - Or in code:

```
v += a * dt
```

    - Note how similar this is to position update!
    - Also note how the units cancel / match.
    - We usually do this velocity update **and** the position update at the same time

Simple Example

- Downwards constant acceleration



Recap

- Do this every frame:

```
pos += vel * dt
vel += accel * dt
```

- Called Newton-Euler-1 integration
  - Terrible, but easy (and common)
  - Runga-Kutta-4, Verlet, etc. integrations are much more accurate.

Newtons 3 laws of motion

- (paraphrased)
- **First Law**: An object at rest stays at rest and an object in motion stays in motion *unless acted upon by a net force.*
- **Second Law**: $\vec{F} = m\vec{a}$
  - Or perhaps more useful to us: $\vec{a} = \frac{\vec{F}}{m}$
- **Third Law**: Whenever a force is applied to body A by body B, an equal-magnitude / opposite-direction force is applied to body B.
- https://en.wikipedia.org/wiki/Newton's_laws_of_motion

Planetary Gravity

- All celestial bodies apply gravitational forces to every other celestial body
  - …in the entire universe! <mind blown>
  - The farther apart they are, the less influence they have
  - More massive objects apply a greater force than less massive objects
- **Newton's Law of Universal Gravitation**

$$\overrightarrow{F_{A\to B}} = G\frac{m_A m_B}{\left\|\overrightarrow{dir_{A\to B}}\right\|^2} * \widehat{dir_{A\to B}}$$

$$G = 6.67408x10^{-11} \quad meters^3 kg^{-1} s^{-2}$$

$$\overrightarrow{F_{B\to A}} = -\overrightarrow{F_{A\to B}}$$

- https://en.wikipedia.org/wiki/Gravitational_constant