Game Physics

Simple Physics Engine

CS 415: Game Development

Professor Eric Shaffer

Source: Millington, Ian. Game Physics Engine Development, Second Edition.

ILLINOIS

Newtonian Physics

Let’s build a simple game physics engine!

- We will animate particles (aka point masses)
- Position is changed by velocity
- Velocity is changed by acceleration
- Forces alter acceleration

- Our physics engine will integrate to compute
  - Position
  - Velocity
- We set the acceleration by applying forces

Force and Mass and Acceleration

- How do we update acceleration when force is applied?
- To find the acceleration due to a force we have

$$\ddot{\mathbf{p}} = \frac{1}{m}\mathbf{f}$$

- So we need to know the inverse mass of the particle
  - You can model infinite mass objects by setting this value to 0

Force: Gravity

- Law of Universal Gravitation

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

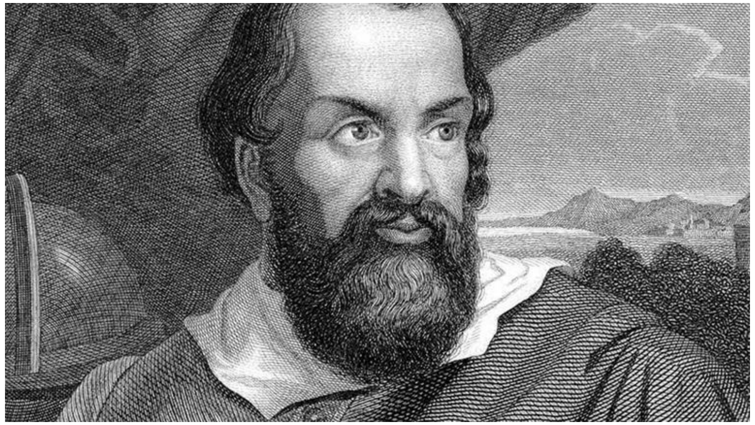
- G is a universal constant
- m_i is the mass of an object
- r is the distance between object centers
- if we care only about gravity of the Earth
    - m1 and r are constants
    - r is about 6400 km on Earth
- We simplify to f = mg
    - g is about $10ms^{-2}$

Acceleration due to Gravity

- If we consider acceleration due to gravity we have

$$\ddot{p} = \frac{1}{m}(mg) = g$$

- So acceleration due to gravity is independent of mass



Acceleration due to Gravity

- Typically the magnitude and direction of acceleration would be

$$\mathbf{g} = \langle 0, -g, 0 \rangle$$

- For gaming, 10ms^{-2} tends to look boring
  - Shooters often use 15ms^{-2}
  - Driving games often use 20ms^{-2}
  - Some tune g object-by-object

Force: Drag

- Drag dampens velocity
  - Caused by friction with the medium the object moves through
- Even neglecting drag, you need to dampen velocity
  - Otherwise numerical errors likely drive it higher than it should be
- A velocity update with drag can be implemented as

$$\dot{\mathbf{p}}_{new} = \dot{\mathbf{p}} d^{t}$$

  - important to incorporate time so drag changes if the frame rate varies

- What range should ***d*** be in?

The Integrator

- The position update can found using Euler’s Method:

$$\mathrm{p}_{new} = \mathrm{p} + \dot{\mathrm{p}}t$$

- This is a pretty inaccurate approximation of analytical integration
  - formula gets more inaccurate as acceleration gets larger...why?
  - In general we can characterize Euler method error as O(t)
  - …almost good enough for game engines...most use semi-implict Euler
- The velocity update is computed using Euler integration as well

$$\dot{\mathbf{p}}_{new} = \dot{\mathbf{p}}d^{t} + \ddot{\mathbf{p}}t$$

The Timestep

How does the timestep effect the accuracy of the engine?

How would you see that error in a game?

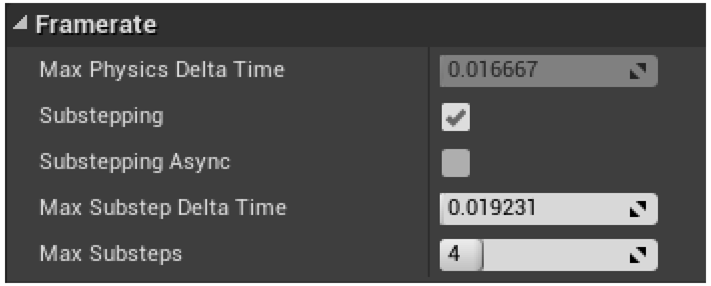
Timestep in games uses wall-clock time (sometimes a scaled version)

- Also related to framerate

If the physics timestep is tied to the framerate, what can happen?

UE Substepping

In UE look at *Project Settings > Engine > Physics*



Framerate

Max Physics Delta Time 0.016667

Substepping ☑

Substepping Async ☐

Max Substep Delta Time 0.019231

Max Substeps 4

- Substepping creates a degree of framerate independence for physics
- When framerate drops, Unreal will add extra physics iterations
- Physics timestep will not exceed the max delta time

Example:

- 16 ms Max Delta Time
- Above 60fps no substepping
- In 30 to 60 fps 2 steps