An introduction to framerate-independent game physics

Anders P. Åsbø
(Dated: 16. november 2022)

CONTENTS

I. INTRODUCTION

A major part of game programming is handling the motion of objects or characters. However, it is often dificult to design authentic feeling game physics and a lot of new game programmers work their way towards usable physics by trial and error. With this paper, I aim to offer an introduction to real-world physics in the form of Newtonian mechanics, as well as the mathematical framework for finding movement from the forces applied. The intention is to help build intuition for how to determine the forces needed when simulating physics, and offer a systematic way to build authentic feeling physics for games.

II. FORCES AND OTHER VECTORS

A. Defining our system

The first step when figuring out the physics of motion, is to define the system we want to simulate the physics of. By system, I refer to the thing that is going to move, be it a ball, swarm, npc or player character. For example, if we want to simulate the physics of a falling ball, then our system will be the ball itself. The ground, air, and any other object that might interact with the ball, are considered external to our system.

III. SLIDING PHYSICS

A. Identifying the forces

Forces involved in a character sliding on a surface are

$$\vec{F_\mu} = -\mu|\vec{N}|\hat{v}_s,$$

where μ is the coefficient of friction (roughness) of the surface, $\vec{N}$ is the normal force

$$\vec{N} = -m\vec{g}\cdot\hat{n},$$

from the surface on the character, where $\hat{n}$ is the unit vector normal to the surface, m is the mass of the character and $\vec{g}$ is the gravitational accelleration vector. Futhermore, $\hat{v}_s$ is the unit vector in the direction of the character's velocity along the surface. It is defined as

$$\hat{v}_s = \vec{v} - (\hat{n}\cdot\vec{v})\,\hat{n},$$

where $\vec{v}$ is the velocity of the character. Finally, the character is affected by gravity

$$\vec{G} = m\vec{g}.$$

B. Finding acceleration from forces, and updating velocity

which can be written mathematically as

$$\sum \vec{F} = m\vec{a},$$

where Σ is shorthand for sum of all"and $\vec{F}$ is shorthand for the forces acting on our system (character). $\vec{a}$ is the acceleration vector that we want to find, since it can be used to calculate per frame change in velocity.

Using the forces defined for our system we get

$$\begin{aligned}\vec{F_\mu} + \vec{G} &= m\vec{a},\\ \vec{a} &= \frac{\vec{F_\mu} + \vec{G}}{m}\\ &= \frac{\vec{F_\mu}}{m} + \vec{g}.\end{aligned}$$

which is the acceleration we're interested in. We can then find the change in velocity $\Delta\vec{v}$ as acceleration multiplied by the time spent accelerating, which per frame is deltaTime Δt. Thus, the updated velocity is the old velocity plus the change in velocity

$$\begin{aligned}\vec{v}_{\text{new}} &= \vec{v}_{\text{old}} + \Delta\vec{v} \\ &= \vec{v}_{\text{old}} + \vec{a}\Delta t.\end{aligned}$$

C. Finding change in position

Now that we have found the change in velocity between frames, we need to find the change in position, which gives us the actual motion of the character. To do this, we use that the change in position over a given time is equal to the updated velocity multiplied by deltaTime

$$\begin{aligned}\vec{r}_{\text{new}} &= \vec{r}_{\text{old}} + \vec{v}_{\text{new}}\Delta t \\ &= \vec{r}_{\text{old}} + \vec{v}_{\text{old}}\Delta t + \vec{a}\Delta t^2.\end{aligned}$$

This is called forward Euler integration, and is the simplest way to calculate movement from forces when the timestep deltaTime is variable. However, it is also one of the least accurate methods, and does not conserve important quantities like total energy and total momentum. Most physics engines use a fixed timestep that does not vary, allowing for more accurate integration methods like Velocity-Verlet, which *does* conserve total energy and total momentum, and therefore garantee that the system won't suddenly gain a lot of kinetic energy and yeet itself to infinity, among other things.