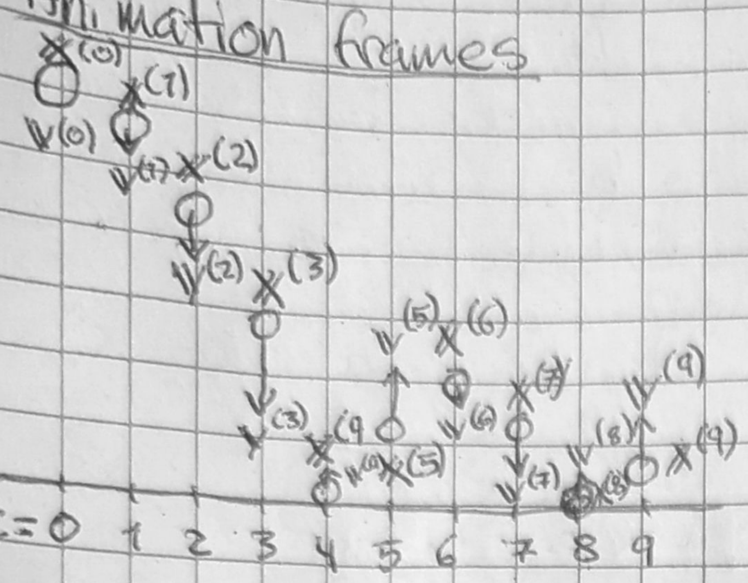
24 - Physics based animation

Animation frames



Simulation state (at time t):

position $x^{(t)}$

velocity $v^{(t)} = \frac{dx^{(t)}}{dt} = \dot{x}^{(t)}$

Given $x^{(t)}, v^{(t)}$

compute $x^{(t+\Delta t)}, v^{(t+\Delta t)}$

Δt is the step time size

Newton's 1st law of motion:

If there is no external force, then $v^{(t+\Delta t)} = v^{(t)}$

$x^{(t+\Delta t)} = x^{(t)} + \Delta t\, v^{(t)}$

Newton's 2nd law of motion:

If there is constant external force, then $F = ma \Leftrightarrow a = F/m$

$a = \dot{v} = \ddot{x}$

$v^{(t+\Delta t)} = v^{(t)} + \Delta t\, a$

$x^{(t+\Delta t)} = x^{(t)} + \Delta t\, v^{(t)} + \frac{1}{2}(\Delta t)^2 a$

If there is varying external force, then $F^{(t)} = ma^{(t)} \Leftrightarrow a^{(t)} = F^{(t)}/m$

Integrals very hard to compute, need to estimate with numerical integration.

$$v^{(t+\Delta t)} = v^{(t)} + \int_t^{t+\Delta t} a^{(t^*)}\, dt^*$$

$$x^{(t+\Delta t)} = x^{(t)} + \int_t^{t+\Delta t} v^{(t^*)}\, dt^*$$

Euler integration (the easiest numerical integration)

We can estimate with these because Δt is often very small.

Explicit:

$a^{(t)} = F^{(t)}/m$

$v^{(t+\Delta t)} = v^{(t)} + \Delta t\, a^{(t)}$

$x^{(t+\Delta t)} = x^{(t)} + \Delta t\, v^{(t)}$

Implicit

$a^{(t+\Delta t)} = F^{(t+\Delta t)}/m$

$v^{(t+\Delta t)} = v^{(t)} + \Delta t\, a^{(t+\Delta t)}$

$x^{(t+\Delta t)} = x^{(t)} + \Delta t\, v^{(t+\Delta t)}$

Here we compute values at the end of the timestep, as opposed to the start. This creates more stable simulations, as ~~we~~ the energy tapers off instead of adding more

Semi-implicit: Like expicit, but $x^{(t+\Delta t)} = x^{(t)} + \Delta t\, v^{(t+\Delta t)}$ because implicit is really hard to calculate.

Forces

gravity force

mass

$\mathbb{F}^G = m\mathbf{g}$ ← gravitational acceleration

Linear spring force: $\boxed{\text{rest length}}$

← $\hat{\mathbf{d}}$

$$\mathbb{F}^S = k(l - l_{rest})\hat{\mathbf{d}}$$

$\boxed{\text{spring force}}$ $\boxed{\text{spring stiffness}}$ $\boxed{\text{spring length}}$ $\boxed{\text{spring direction}}$

The spring will oscillate indefinitely, so we need a ~~damp~~ damping force: $\mathbb{F}^D = k\dot{l}\hat{\mathbf{d}}$

$\boxed{\text{damping force}}$ $\boxed{\text{damping stiffness}}$ $\boxed{\text{length change speed}}$ $\boxed{\text{spring direction}}$

Mass-spring systems

$\mathbb{F}_0^s$ $\qquad \mathbb{F}_1^s = -\mathbb{F}_0^s$ ← spring force: $\mathbb{F}_0^s = k(l - l_{rest})\hat{\mathbf{d}}$

$\mathbf{x}_0 \mathbf{v}_0 \qquad \mathbf{x}_1 \mathbf{v}_1$

$l = |\mathbf{x}_1 - \mathbf{x}_0|$

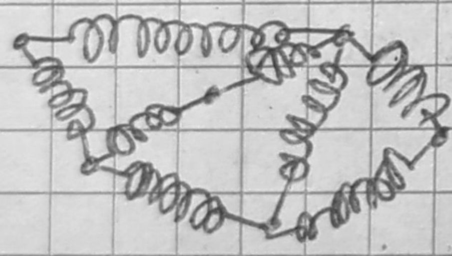
$\hat{\mathbf{d}} = (\mathbf{x}_1 - \mathbf{x}_0)/l$

$\mathbb{F}_0^D \qquad \mathbb{F}_1^D = -\mathbb{F}_0^D$ ← Damping force: $\mathbb{F}_0^D = k\dot{l}\hat{\mathbf{d}}$

$\mathbf{x}_0 \mathbf{v}_0 \qquad \mathbf{x}_1 \mathbf{v}_1$

$\dot{l} = (\mathbf{v}_1 - \mathbf{v}_0) \cdot \hat{\mathbf{d}}$

$\hat{\mathbf{d}} = (\mathbf{x}_1 - \mathbf{x}_0)/l$



we can have complex mass-spring systems with n mass particles

their / simulation steps

positions: $\mathbf{x} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}\}$

velocities: $\mathbf{v} = \{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n-1}\}$

Simulation pseudocode:

- Initialize state
- For each time step
  - compute simulation step
  - Display/record new state

Simulation step pseudocode:

Compute total force on each particle:

$\mathbf{f} = \{\mathbf{f}_0, \mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n-1}\}$

For each particle i:

$a_i^{(t)} = f_i^{(t)}/m_i$

Calculate velocity:

$\mathbf{v}_i^{(t+\Delta t)} = \mathbf{v}_i^{(t)} + \Delta t\, a_i^{(t)}$

Calculate position:

$\mathbf{x}_i^{(t+\Delta t)} = \mathbf{x}_i^{(t)} + \Delta t\, \mathbf{v}_i^{(t+\Delta t)}$

semi-implicit euler

Force computation pseudocode:

Initialize: $\mathbf{f} = \{0, 0, \ldots\}$

For each particle i:

  Add gravity: $\mathbf{f}_i = \mathbf{f}_i + m_i \mathbf{g}$

For each spring between particle i and j

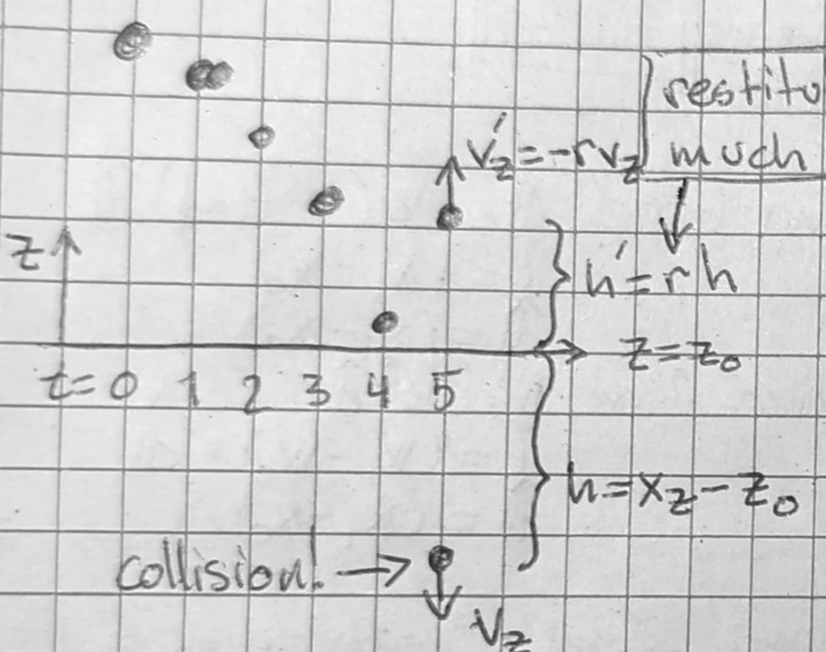
  Compute spring forces $\mathbf{f}_i^s$ and $\mathbf{f}_i^D$

  Add spring force:

$$\mathbf{f}_i \leftarrow \mathbf{f}_i + (\mathbf{f}_i^s + \mathbf{f}_i^D)$$

$$\mathbf{f}_j \leftarrow \mathbf{f}_j - (\mathbf{f}_i^s + \mathbf{f}_i^D)$$

Collisions



restitution coefficient, estimates how much energy is lost in the collision.

This is a simpel model, can also replace the particle at z_0.

Also in 3D, we need to account for all axes.