Recap

Let x(t) be the **position** of the center of mass of the rigid body

Let v(t) be the **velocity** of the rigid body

Let m be the **mass** of the rigid body. m is a scalar

P(t) is the **linear momentum** of the body, P(t) = mv(t)

➔ $\dot{x}(t)$ = d/dt(x(t)) = v(t), F = ma = m(d/dt(v)) = d/dt(P(t))

➔ $\dot{P}(t)$ = F

These two equations describe **linear motion**

$\omega(t)$ – **Angular velocity** of the body

q(t) – **Orientation** of the body (using **unit quaternions**)

$I(t)_{3x3}$ – The **Inertia tensor**

Obj Space/World Space Transformations

$I_{world}(t) = R^T(t)I_{body}R(t)$

$L(t) = I(t)\,\omega(t)$, $d/dt(L(t)) = \dot{L}(t) = \tau$, where τ is the applied torque

➔ To numerically compute these values, we need to discretize time

Time Discretization

$\dot{P}(t) = F \rightarrow (P^{n+1} - P^n) / \triangle t = F$, this is the **Forward Euler**

$v^{n+1} = P^{n+1}/m \rightarrow \dot{x} = v \rightarrow (x^{n+1} - x^n) / \triangle t = v^{n+1}$, the **Backward Euler**

➔ $= x^n + \triangle t(v^{n+1})$

For the angular part,

$\dot{L}(t) = \tau = (L^{n+1} - L^n) / \triangle t$, $L^{n+1} = L^n + \triangle t * \tau$

➔ $L = Iw \rightarrow w^{n+1} = I^{-1}L^{n+1}$, I is the world space inertia tensor

➔ $R(t)I^{-1}_{object}R(t)^T$

The object's inertia tensor never changes. Inversion can be expensive, so if we choose a principle axis, we can use a diagonal matrix, so the **inverse of a diagonal matrix** is simply **1 over each diagonal entry** in the diagonal matrix, so this is computationally efficient.

Unit Quaternion

q^{n+1}, a unit quaternion, is a 4x1 vector

$\dot{q}(t) = \frac{1}{2}(\omega(t)_{3x1}q(t))_{4x1}$, this formula is extremely deceptive.

$\dot{R}(t) = \omega(t)*R(t)$

We can assume that $\vec{\omega} = \begin{matrix} w1 \\ w2 \\ w3 \end{matrix}$, $\omega^* = \begin{matrix} 0 & -w3 & w2 \\ w3 & 0 & -w1 \\ -w2 & w1 & 0 \end{matrix}$, which is a

skew-symmetric matrix where $A_{ij} = -A_{ji}$

The Cross Product of two 3x1 vectors is: $\vec{a}_{3x1} \times \vec{b}_{3x1} = \vec{c}_{3x1}$

We can look at this as: $[\vec{a}_x]_{3x3}[\vec{b}]_{3x1} = [\vec{c}]_{3x1}$, so it makes sense that

$\dot{R}(t)_{3x3} = \omega(t)_{3x3}*R(t)_{3x3}$, $(R^{n+1} - R^n) / \triangle t = (\omega^{n+1} \times R^n)_{3x3}$

1. The full matrix makes updating easier, so we can covert R^{n+1} back to q^{n+1}, assuming that R^n is orthonormal, so $R^T T = I$, which allows $R^{n+1} = q^{n+1}$, the unit quaternion.
2. You must ortho-normalize R^{n+1} to be sure that the new transform corresponds to axes perpendicular to it.
    a. If not, the object will stretch to infinite, which is not a rigid body.
    b. Alternatively, this property may be used for NPR – Non Photorealistic Rendering.

3. Bullet Physics allows you to write code in python to call efficient c++ code using an open dynamics engine.

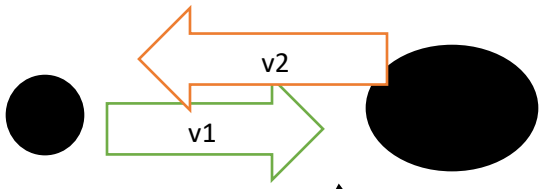
Collision Detecting

This is separated into two categories because there are two effects.

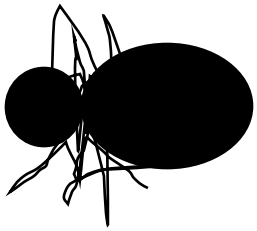
1. Separating (non-persistent collision)

   For example:

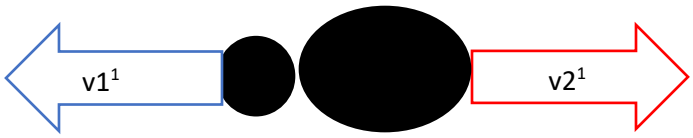
   Stage 1:



   Stage 2:



   Stage 3:

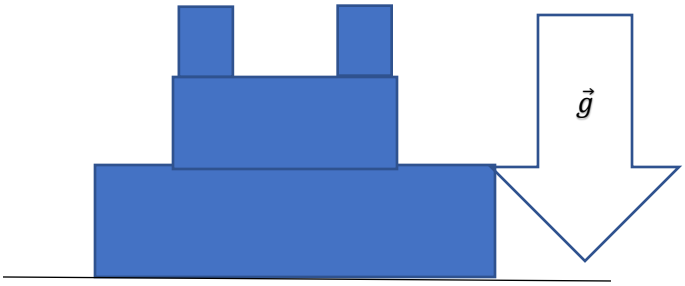


   Because of the conservation of momentum, physics can help us understand what goes on with colliding objects.

   a. If two objects of similar size collide, they tend to bounce away from each other at similar speeds

b. If two objects if differing size collide, the larger object tends to bounce away much slower speed than the smaller object, or the objects move in tandem.

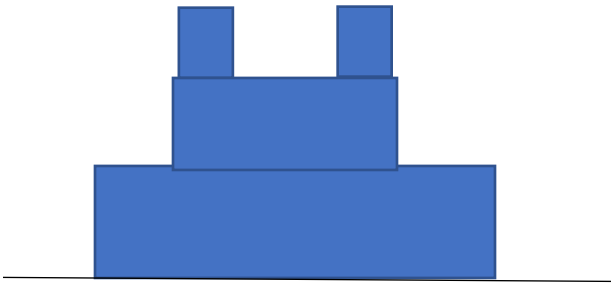
2. Resting contact (persistent collision)



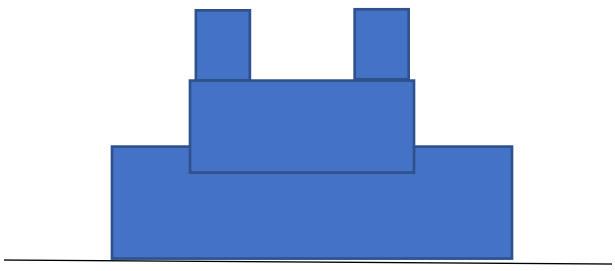
Gravity is acting on all objects.

a. Assume that the ground is still, but the object can move

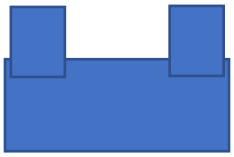
i. First object interpolation



ii. Second object interpolation



iii. Third object interpolation



iv. And Finally