3.1 Equation of Motion

$i = [m_i, x_i, v_i]$

$f_i = a_i m_i = \dot{v}_i m_i \qquad \dot{x}_i = v_i$

sum of all force apply on particle i

Also inertial tensor $I_i \in R^{3\times3}$ 角动量

$$I_i = \begin{bmatrix} I_{xx}, & I_{xy}, & I_{xz} \\ I_{yx}, & I_{yy}, & I_{yz} \\ I_{zx}, & I_{zy}, & I_{zz} \end{bmatrix} \Rightarrow H = I\omega$$

Angular velocity

inertial tensor

Generally choose mass center as origin, axis to make I_i a diagonal matrix.

Rotation motion equation

$$\dot{\omega}_i I_i = \tau_i - (\omega_i \times (I_i \omega_i)) \xrightarrow{\text{proof}}$$

sum of moment

By Euler second law

$$M_i = \frac{dL_i}{dt}$$

$$L_i = I_i \cdot \omega_i$$

$$M_i = \dot{I}_i \omega + I\dot{\omega}$$

$$M_i = \left(\frac{dL}{dt}\right)_{rot} + \omega \times L$$

$$M_i = I_i\dot{\omega}_i + \omega_i \times (I_i \omega_i)$$

velocity kinematic relation

$$\dot{q}_i = \frac{1}{2}\hat{\omega}_i q_i$$

3.2 time integrate

In contrast to the well-known explicit Euler, the symplectic Euler uses the velocity at time $t_0 + \Delta t$ instead of time t_0 for the integration of the position vector. The time integration for a particle is then performed by the following equations:

$$\begin{aligned} \mathbf{v}_i(t_0+\Delta t) &= \mathbf{v}_i(t_0) + \Delta t \frac{1}{m_i}\mathbf{f}_i(t_0) \\ \mathbf{x}_i(t_0+\Delta t) &= \mathbf{x}_i(t_0) + \Delta t\, \mathbf{v}_i(t_0+\Delta t). \end{aligned}$$

simple $V_{next} = V_{cur} + a \cdot dt$

$X_{next} = X_{cur} + v \cdot dt$

In the case of a rigid body also Equations (3) and (4) must be integrated. Using the symplectic Euler method this yields:

$$\begin{aligned} \boldsymbol{\omega}_i(t_0+\Delta t) &= \boldsymbol{\omega}_i(t_0) + \Delta t \left(\mathbf{I}_i^{-1}(t_0) \cdot (\boldsymbol{\tau}_i(t_0) - (\boldsymbol{\omega}_i(t_0) \times (\mathbf{I}_i(t_0)\boldsymbol{\omega}_i(t_0)))) \right) \Rightarrow \dot{\omega} \\ \mathbf{q}(t_0+\Delta t) &= \mathbf{q}(t_0) + \Delta t \frac{1}{2}\tilde{\boldsymbol{\omega}}_i(t_0+\Delta t)\mathbf{q}_i(t_0). \end{aligned}$$

Can have bette way for approach than Euler.

3.3 constrain

kinematic constrain on x, v, w

$$C(x_{i_1}, q_{i_1}, \cdots, x_{i_{n_j}}, q_{i_{n_j}}) = 0$$
$$\geq 0$$

$\{i_1, \cdots i_{n_j}\}$ is body indice

n_j is cardinality of constrain

4. The Core of position based dynamic

4.1 Algorithm.

object: N particles

M constrains $\leftarrow k_m$ for each constrain (stiffness parameter)

$k \in [0, 1]$

4.1.1.

Algorithm 1 Position-based dynamics

for all vertices i **do**
  initialize $\mathbf{x}_i = \mathbf{x}_i^0$, $\mathbf{v}_i = \mathbf{v}_i^0$, $w_i = 1/m_i$ → init all particles
end for
loop
  for all vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{\mathrm{ext}}(\mathbf{x}_i)$ } → model prediction
  for all vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
  for all vertices i **do** genCollConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$) → generate constrain
  loop solverIteration **times**
    projectConstraints($C_1, \ldots, C_{M+M_{\mathrm{Coll}}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$) → solve constrain to update p_i
  end loop
  for all vertices i **do**
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$ } → use updated p_i for v_i and x_i
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  end for
  velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
end loop

4.1.2 Damping

Damping is used to make simulation stable. Reduce the temporal oscillation.

$\Rightarrow Cx$

C is demping matrix.