7.4 Rigid body simulation*

Objective: create ***realistic-looking motion*** for physically based reaction of rigid bodies to forces such as gravity, viscosity, friction, and those resulting from collisions with ***key-frame techniques***

Covers two parts:

-***unconstrained* motion**: simulations that aren't concerned about collisions between rigid bodies

- ***constrained* motion**: regard bodies as solid, and need to disallow inter-penetration

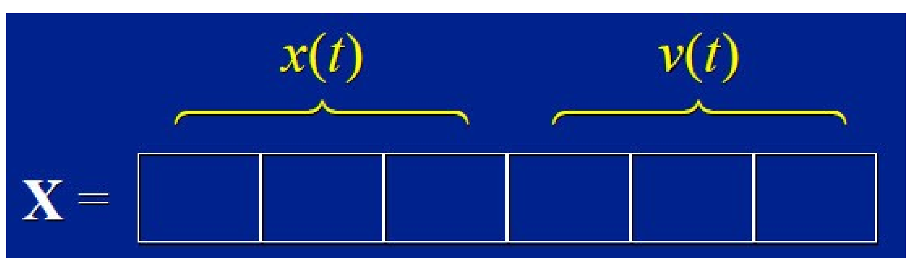
• *Some materials used here are taken from David Baraff's notes:* **Physically Based Modeling - *Rigid Body Simulation***

7.4.1 Unconstrained Rigid Body Dynamics

Simulation basics

- basic structure for simulating the motion of a rigid body
- (almost) the same as simulating the motion of a particle
- *x(t):* particle's location in world space at time *t*
- $v(t) = \dot{x}(t) = \frac{d}{dx}x(t)$: velocity of the particle at time *t*
- state vector X(t) of a particle at time t is the particle's position and velocity

$$\mathbf{X}(t) = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} \qquad (7\text{-}1)$$



7.4.1 Unconstrained Rigid Body Dynamics

Simulation basics (conti)

- For system with *n* particles, enlarge X(t) to be

$$X(t) = \left(x_1(t), v_1(t), \cdots, x_n(t), v_n(t)\right)^T$$

- *F(t) :* force acting on particle at time t, sum of all forces acting on particle: gravity, wind, spring forces, etc.

- If particle *i* has mass m_i, then change of X over time is given by

$$\frac{d}{dt}\mathbf{X} = \frac{d}{dt}\begin{pmatrix} x_1(t) \\ v_1(t) \\ \vdots \\ x_n(t) \\ v_n(t) \end{pmatrix} = \begin{pmatrix} v_1(t) \\ F_1(t)/m_1 \\ \vdots \\ v_n(t) \\ F_n(t)/m_n \end{pmatrix} \qquad (7\text{-}2)$$

$$\frac{d}{dt}\mathbf{X} = \begin{array}{|c|c|c|c|c|} \hline \quad & \quad & \cdots\ 6n \text{ elements}\ \cdots & \quad & \quad \\ \hline \end{array}$$

7.4.1 Unconstrained Rigid Body Dynamics

Simulation basics (conti)

- given any value of *X(t)*, equation (7-2) describes how *X(t)* is instantaneously changing at time t
- A simulation starts with initial conditions for X(0) (values for x(0) and v(0)) and then uses an ode solver to track the change ("flow") of X(t), for as long as we're interested in. To animate the motion of the particle, compute *X(1/30), X(2/30) ...*
- how we'd actually interact with a numerical solver (ode), in a C++-like language

Numerical solver

```
typedef void (*DerivFunc)(double t, double x[], double xdot[]);

void    ode(double x0[], double xEnd[], int len, double t0,
                double t1, DerivFunc dxdt);
```

Simulation basics (conti)

Numerical solver

```
typedef void (*DerivFunc)(double t, double x[], double xdot[]);

void    ode(double x0[], double xEnd[], int len, double t0,
            double t1, DerivFunc dxdt);
```

x0: initial state vector to ode

len: length of x0

t0, t1: starting and ending times of simulation

xEnd: state vector at t1 returned by ode

dxdt( ): a function passed to ode; given an array y that encodes a state vector *X(t)* and a time *t, dxdt() computes and returns* $\frac{d}{dt}X(t)$ in the array *xdot;* ode is allowed to call *dxdt* as often as it likes.

Rigid Body Concepts

- simulating rigid bodies is like simulating particles, except more complicated state vector *X(t)* and derivative $\frac{d}{dt}X(t)$
- use the same paradigm of tracking the movement of a rigid body using a solver *ode*, with a provided *dxdt()*
- to describe the motion of a rigid body, one needs

  x(t): describes translation of the body

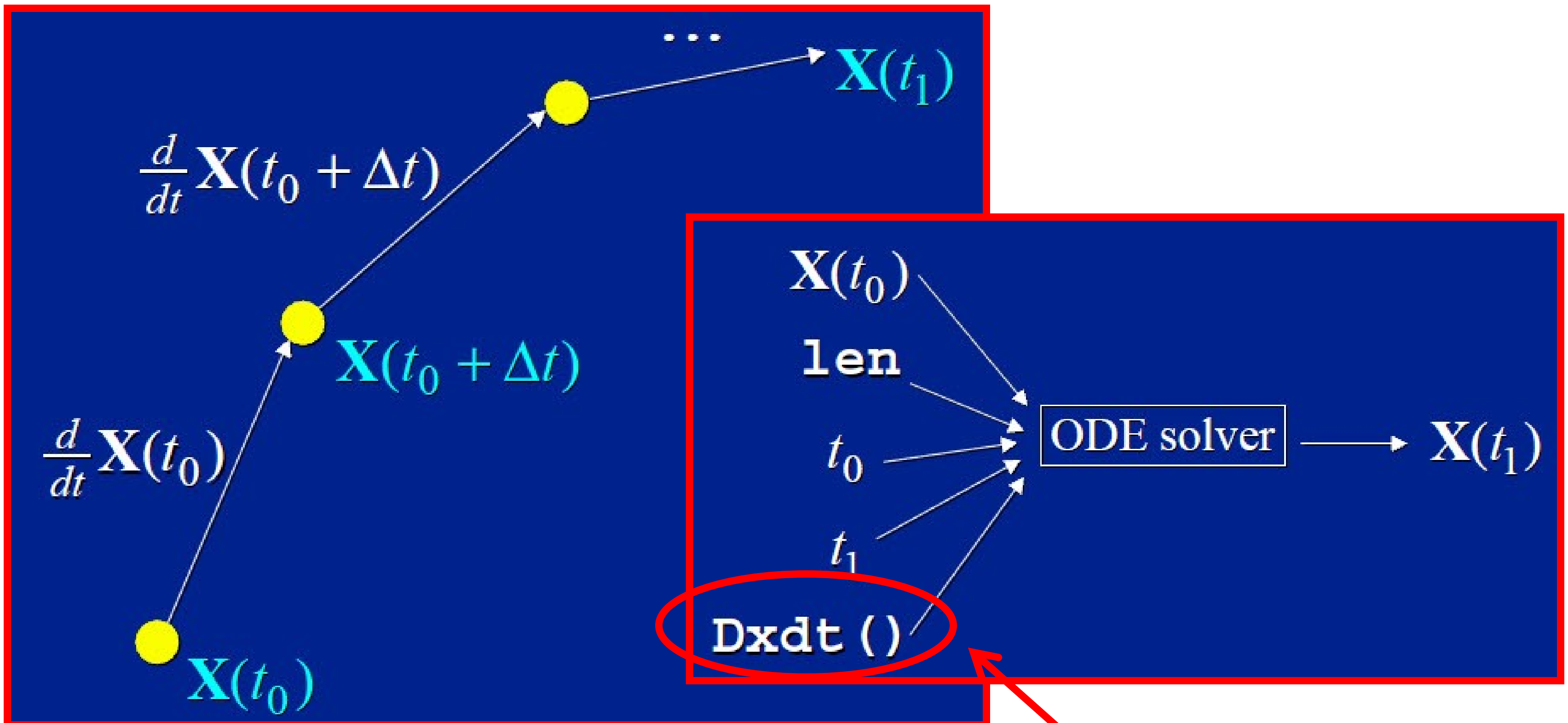
  R(t): describes rotation of the body

  spatial variables

  quaternions?
- the rigid is defined in a *body space* (fixed & unchanged local space; mass center of the body lies at the origin)
- geometric description of the body in body space is transformed into *world space* by *x(t) and R(t)*

Rigid Body Concepts

- movement tracking of a rigid body using *ode*, with a provided *dxdt( )*



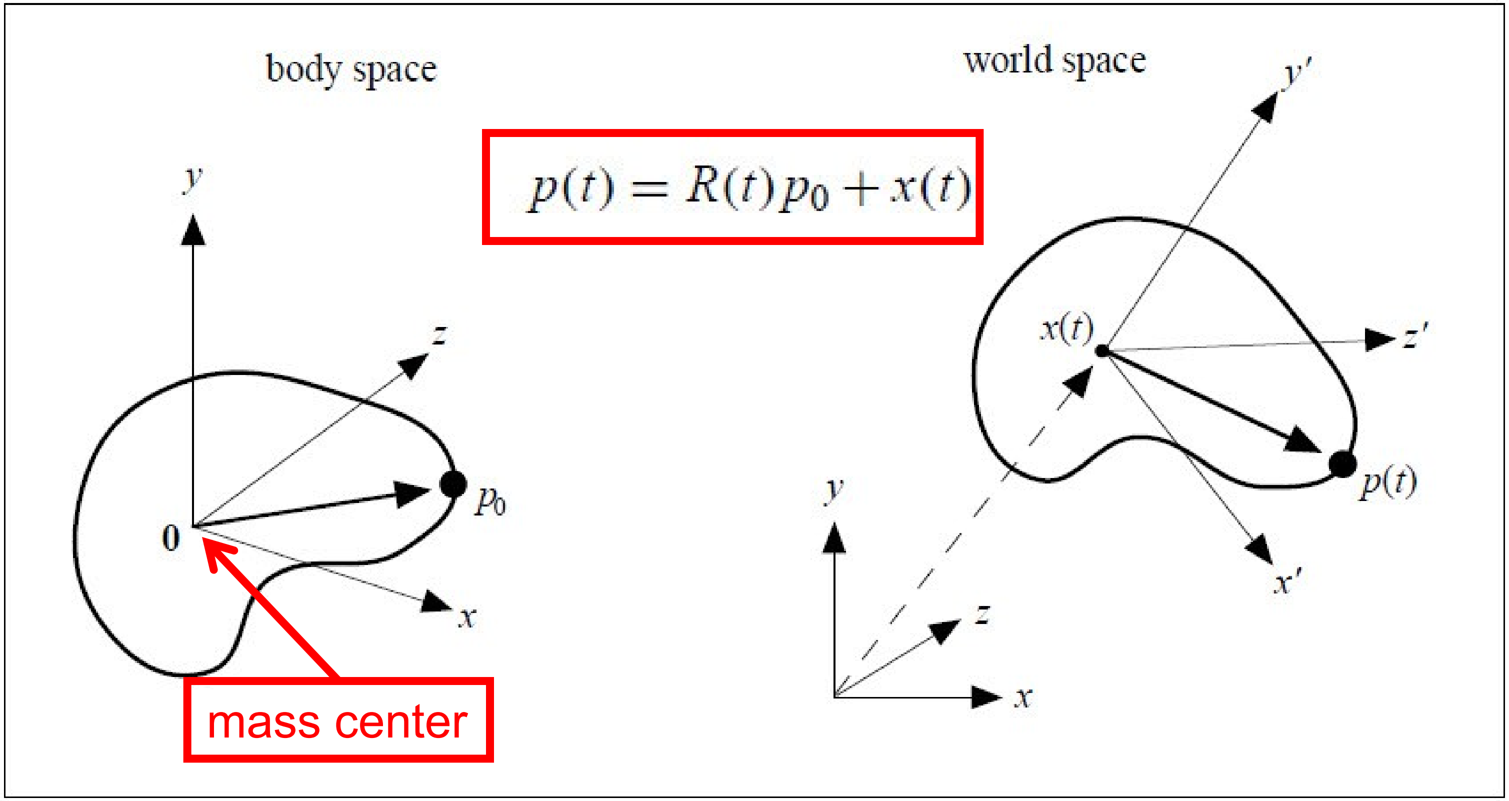

```
void Dxdt(double t, double x[],
                    double xdot[])
```

Rigid Body Concepts

- provided *dxdt( )*

```
void Dxdt(double t, double x[],
                    double xdot[])
```

$$\mathbf{X}(t) = \begin{bmatrix} x_1(t) \\ v_1(t) \\ \vdots \\ x_n(t) \\ v_n(t) \end{bmatrix} \qquad \frac{d}{dt}\mathbf{X}(t) = \begin{bmatrix} v_1(t) \\ F_1(t)/m_1 \\ \vdots \\ v_n(t) \\ F_n(t)/m_n \end{bmatrix}$$



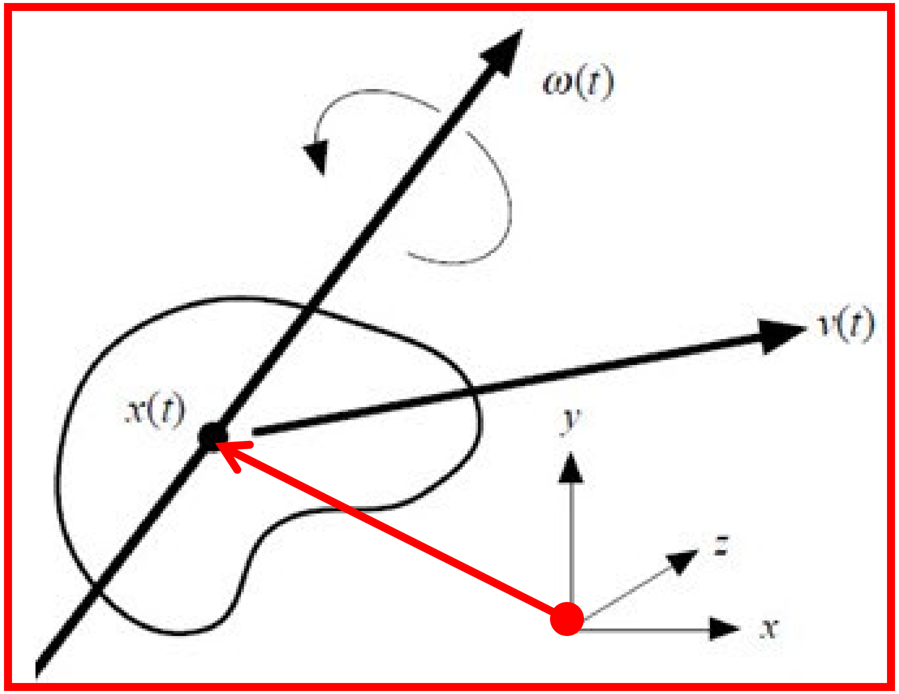
$$R(t) = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix}$$

$$= (x', y', z')$$

R's first column gives the direction that the rigid body's *x axis points in, when transformed to world space at time t*

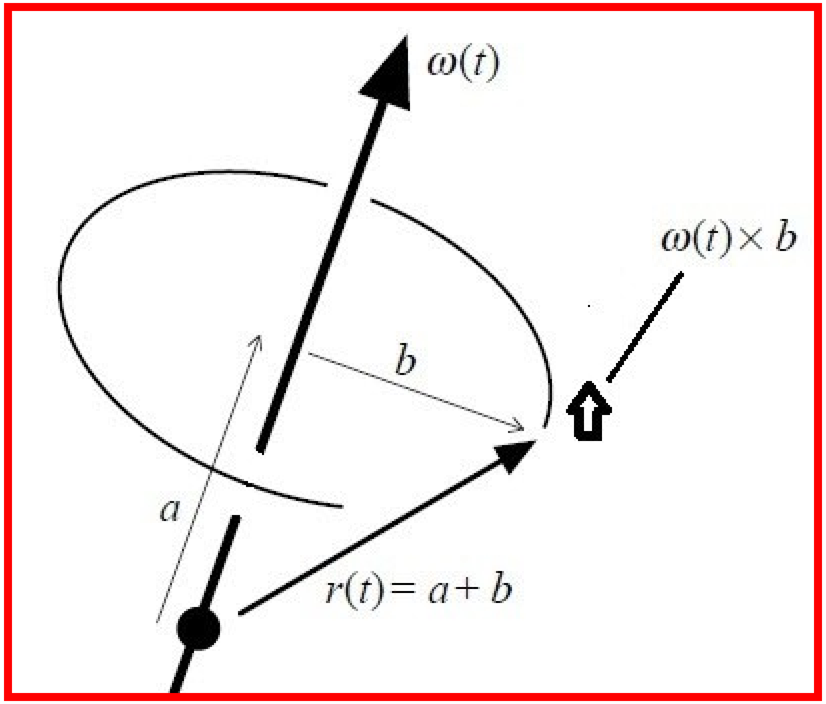
Velocities (linear and angular)

- define how the position and orientation change over time
- a rigid can translate and spin
- need $\dot{x}(t)$ and $\dot{R}(t)$
- linear velocity $v(t) = \dot{x}(t)$
- angular velocity $\omega(t)$:
  a vector, encodes both the axis of the spin and the speed of the spin
- How are R(t) and $\omega(t)$ related?



Velocities (linear and angular)

- how the change of an arbitrary vector in a rigid body is related to the angular velocity $\omega(t)$



r(t), fixed to the rigid body;
as a direction, independent of any translational effects, in particular, $\dot{r}(t)$ is independent of $v(t)$

Assumption: the rigid body were to maintain a constant angular velocity

Conclusion: the tip of *r(t)* traces out a circle centered on the $\omega(t)$ axis ;
instantaneous velocity of *r(t)* has magnitude $|b||\omega(t)|$

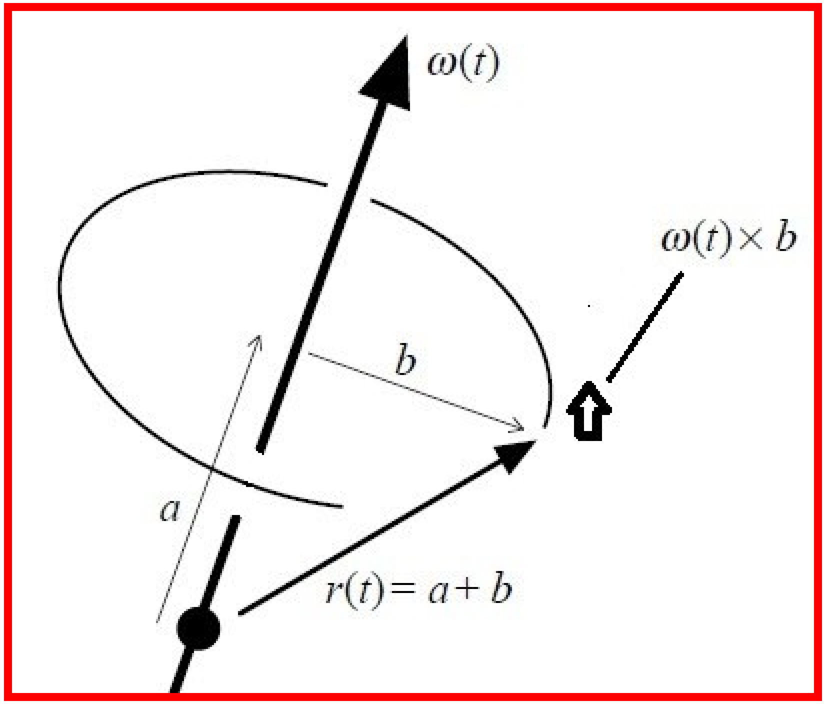
Velocities (linear and angular)

On the other hand, we have

$$|\omega(t)\times b| = |b||\omega(t)|$$

Consequently, we have

$$\begin{aligned}\dot{r}(t) &= \omega(t)\times b \\ &= \omega(t)\times b + \omega(t)\times a \\ &= \omega(t)\times(b+a) \\ &= \omega(t)\times r(t)\end{aligned}$$



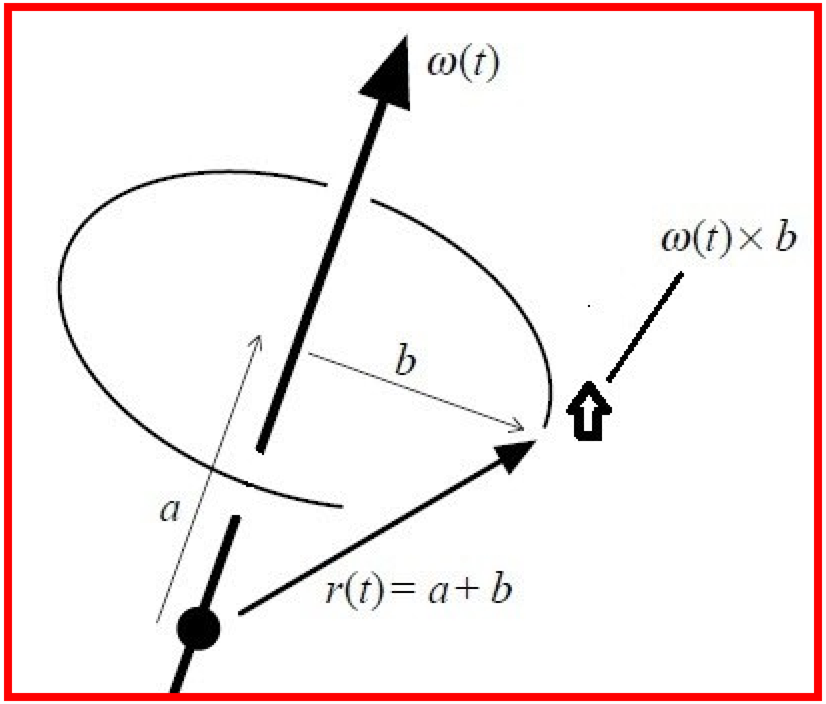
Put all this together:

(1) At time *t,* the direction of the x axis of the rigid body in world space is the first column of R(t):

$$\begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix}$$

Velocities (linear and angular)

(2) At time *t,* derivative of the first column of R(t) is just the rate of change of this vector; using the cross product rule we just discovered, this change is



$$\omega(t) \times \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix}$$

(3) The same holds for the other two columns of *R(t).* This means that we can write

$$\dot{R} = \left(\omega(t) \times \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad \omega(t) \times \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \quad \omega(t) \times \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right)$$

Velocities (linear and angular)

(3) The same holds for the other two columns of *R(t)*. This means that we can write

$$\dot{R} = \left(\omega(t) \times \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad \omega(t) \times \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \quad \omega(t) \times \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right)$$

(4) Note if *a* and *b* are 3-vectors, then $a \times b$ is the vector

$$\begin{pmatrix} a_y b_z - b_y a_z \\ -a_x b_z + b_x a_z \\ a_x b_y - b_x a_y \end{pmatrix}$$

Given the vector *a*, let us define a^* to be the matrix

$$\begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix}$$

anti-symmetric

Velocities (linear and angular)

Then

$$a^*b = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix} \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_y b_z - b_y a_z \\ -a_x b_z + b_x a_z \\ a_x b_y - b_x a_y \end{pmatrix} = a \times b$$

(5) Using the " * " notation, we can rewrite $\dot{R}(t)$ as

$$\dot{R}(t) = \left(\omega(t)^* \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad \omega(t)^* \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \quad \omega(t)^* \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right)$$

or

$$\dot{R}(t) = \omega(t)^* \left(\begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \quad \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right)$$

or simply $\dot{R}(t) = \omega(t)^* R(t)$ (7-3)

Mass of a body

- assume a rigid body is made up of large number of small particles (to make subsequent derivations simpler)

- Notations

m_i : mass of i-th particle (i = 1, …, N)

r_{0i} : location of i-th particle in body space

r_i : location of i-th particle in world space

M : total mass of the body

- Formulas

$$r_i = R(t)\, r_{0i} + x(t) \qquad (7\text{-}4)$$

$$M = \sum_{i=1}^{N} m_i$$

Velocity of a particle

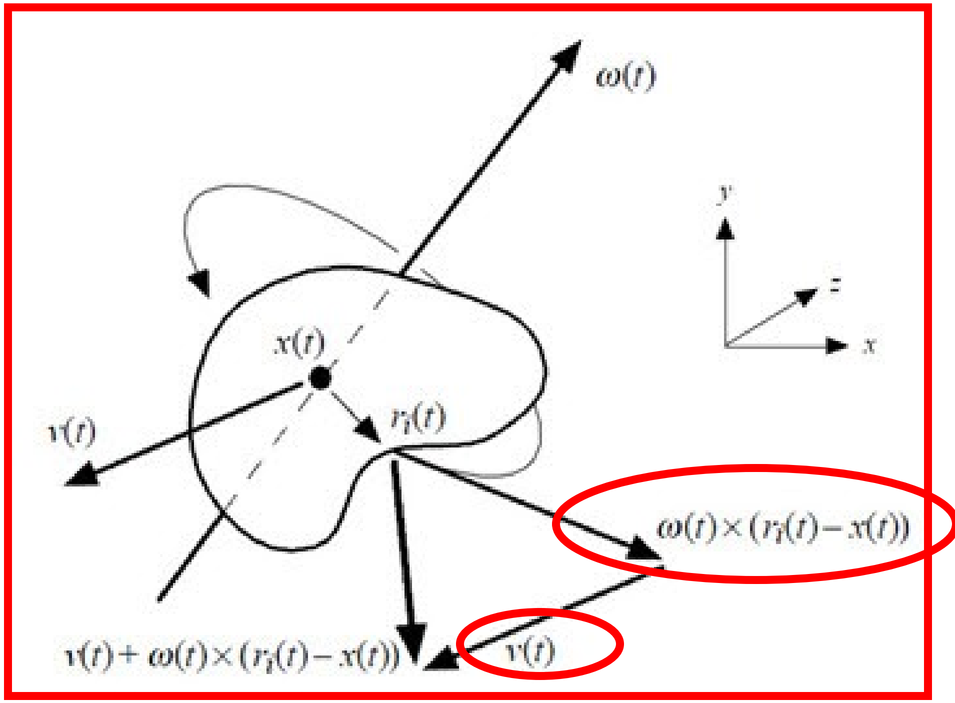
- differentiating (7-4) and using (7-2) to get

$$\dot{r}_i(t) = \omega(t)^* R(t)\, r_{0i} + v(t)$$

- the velocity can be decomposed into a linear term and a angular term

$$\begin{aligned}\dot{r}_i(t) &= \omega(t)^* (R(t)\, r_{0i} + x(t) - x(t)) + v(t) \\ &= \omega(t)^* (r_i(t) - x(t)) \\ &\qquad + v(t) \\ &= \omega(t) \times (r_i(t) - x(t)) \\ &\qquad + \mathrm{v(t)}\end{aligned}$$

……..(7-5)



Center of mass

- enables us to separate the dynamics of bodies into linear and angular components

$$\text{center of mass} \equiv \left(\sum m_i r_i(t)\right) / M$$

- in a center of mass coordinate system for body space, we have

$$\left(\sum m_i r_{0i}\right) / M = \vec{0} = (0, 0, 0)^T$$

- $x(t)$ is the location of the center of mass at time t

$$x(t) = \left(\sum m_i \, r_i(t)\right) / M$$

Why?

$$\frac{\sum m_i r_i(t)}{M} = \frac{\sum m_i (R(t) r_{0i} + x(t))}{M} = \frac{R(t) \sum m_i r_{0i} + \sum m_i x(t)}{M}$$

$$= x(t) \frac{\sum m_i}{M} = x(t)$$

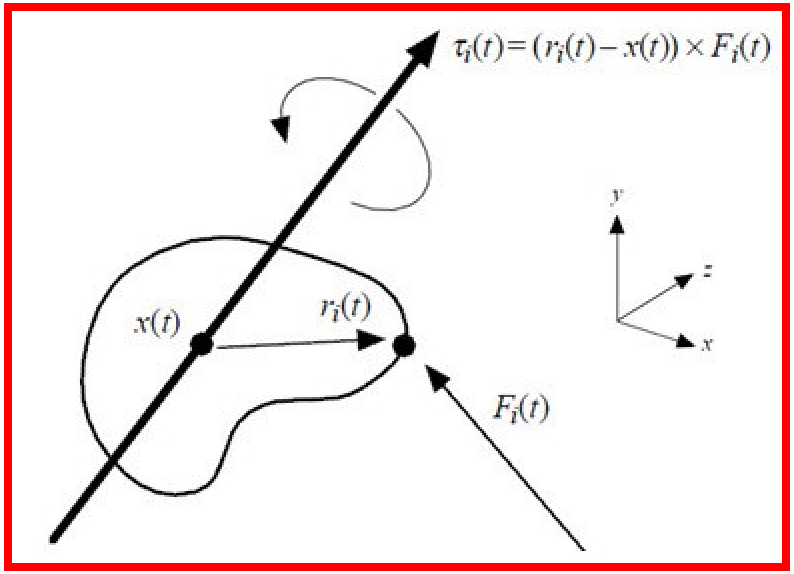
Force and Torque

- $F_i(t)$: total force from external forces acting on the *i-th* particle at time *t.*

- $\tau_i(t)$: external *torque* acting on the *i-th* particle

$$\tau_i(t) = (r_i(t) - x(t)) \times F_i(t)$$

- think of the direction of $\tau_i(t)$ *as* being the axis the body would spin about due to $F_i(t)$



- *F(t)* : total external force

$$F(t) = \sum F_i(t)$$

- $\tau(t)$: total external torque

$$\tau(t) = \sum \tau_i(t) = \sum (r_i(t) - x(t)) \times F_i(t)$$

Linear momentum

- $p_i(t)$: linear momentum of particle m_i with velocity $\dot{r}_i(t)$

$$p_i = m_i\, \dot{r}_i(t)$$

- $P(t)$: total linear momentum

$$\begin{aligned}
P(t) &\equiv \sum m_i\, \dot{r}_i(t) \\
&= \sum \left(m_i v(t) + m_i \omega(t) \times \left(r_i(t) - x(t)\right)\right) \\
&= \sum m_i v(t) + \omega(t) \times \sum m_i \left(r_i(t) - x(t)\right) \\
&= v(t) \sum m_i \\
&= M\, v(t)
\end{aligned}$$

(7-5)

$$\begin{aligned}
&\sum m_i (r_i(t) - x(t)) \\
&= \sum m_i (R(t) r_{0i} + x(t) - x(t)) \\
&= R(t) \sum m_i r_{0i} = 0
\end{aligned}$$

(7-6)

- Consequently,

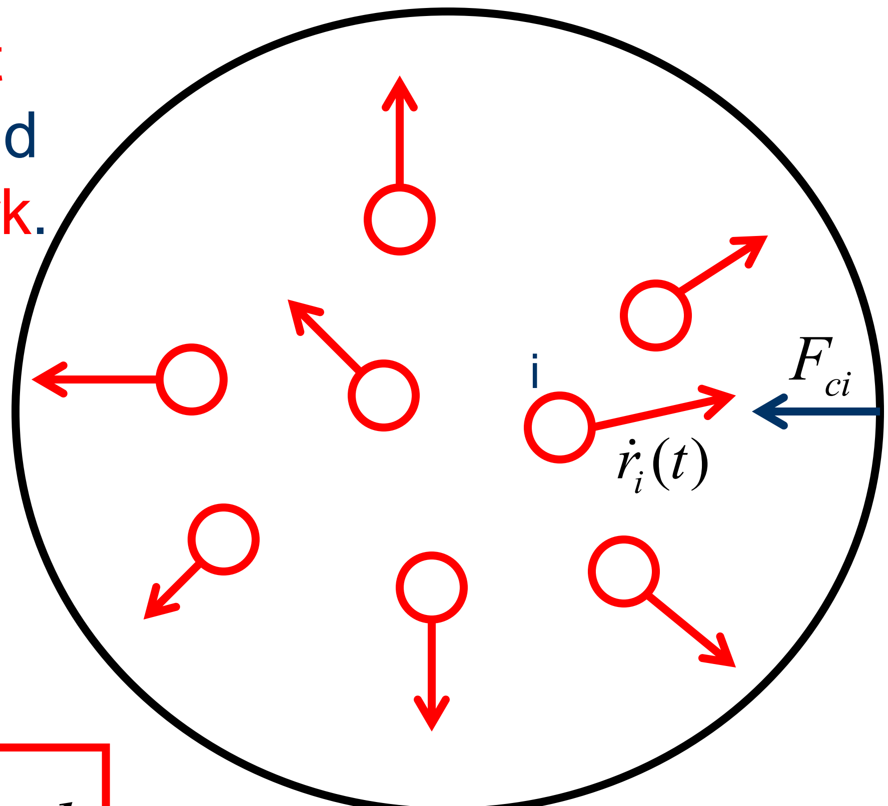
$$\dot{v}(t) = \frac{\dot{P}(t)}{M} = \frac{F(t)}{M}$$

$$\boxed{\dot{P}(t) = F(t) \qquad \text{Why?}}$$

Why is $\boxed{\dot{P}(t) = F(t)}$?

Proof: For a rigid body to maintain its shape, there must be some "internal" constraint forces that act between particles in the same body.

These constraint forces act passively on the system and do not perform any net work.



Let $F_{ci}(t)$ *denote* the net internal constraint force acting on the *i-th particle.* The work performed by F_{ci} on the *i-th* particle from Time t_0 to t_1 is

$$\boxed{\int_{t_0}^{t_1} F_{ci}(t) \cdot \dot{r}_i(t)\, dt}$$

where $\dot{r}_i(t)$ is the velocity of *i-th* particle.

Proof: (conti)

The net work over all the particles is the sum

$$\sum_i \int_{t_0}^{t_1} F_{ci}(t) \cdot \dot{r}_i(t)\,dt = \int_{t_0}^{t_1} \sum_i F_{ci}(t) \cdot \dot{r}_i(t)\,dt$$

which must be zero for any interval t_0 to t_1.

This means that the integrand

$$\sum_i F_{ci}(t) \cdot \dot{r}_i(t) \qquad (7\text{-}7)$$

is itself always zero for any time *t*.

We can use this fact to eliminate any mention of F_{ci} from our derivations. First, some notes about the " * " operator. since *a*b = a x b, and a x b =−b x a, we get*

$$-a^*b = b \times a = b^*a$$

Since *a** is an anti-symmetric matrix, $(a^*)^T = -a^*$

Proof: (conti)

Finally, since the " * " operator is a linear operator, we have

$$(\dot{a})^* = (\dot{a^*}) = \frac{d}{dt}(a^*) \qquad \sum a_i^* = \left(\sum a_i\right)^*$$

for a set of vectors a_i.

Recall that we can write the velocity $\dot{r}_i$ as $v + \omega \times (r_i - x)$ where r_i is the particle's location, x is the position of the center of mass, and v and ω are linear and angular velocity. Letting $r_i' = r_i - x$ and using the " * " notation,

$$\dot{r}_i = v + \omega^* r_i' = v - r_i'^* \omega.$$

Substituting this into (7-7), which is always zero, yields

$$\sum F_{ci} \cdot (v - r_i'^* \omega) = 0.$$

Note that this equation must hold for arbitrary values of v and ω. Since v and ω are completely independent, if we

Proof: (conti)

choose ω to be zero, then $\sum F_{ci} \cdot v = 0$ for any choice of *v*, from which we conclude that in fact $\sum F_{ci} = \boldsymbol{0}$ is always true. This means that the constraint forces produce no net force

Similarly, choosing *v* to be 0 we see that $\sum -F_{ci} \cdot (r_i'^* \omega) = 0$ for any ω . Rewriting $F_{ci} \cdot (r_i'^* \omega)$ as $F_{ci}{}^T (r_i'^* \omega)$ we get that

$$\sum -F_{ci}{}^T r_i'^* \omega = \left(\sum -F_{ci}{}^T r_i'^* \right) \omega = 0$$

for any ω , so $\sum -F_{ci}{}^T \; r_i'^* = 0^{\mathrm{T}}$. Transposing, we have

$$\sum -(r_i'^*)^T F_{ci} = \sum (r_i')^* F_{ci} = \sum r_i' \times F_{ci} = \mathbf{0}$$

which means that the internal forces produce no net torque.

We can use the above to derive the rigid body equations of motion. The net force on each particle is the sum of the internal constraint force F_{ci} *and the external force* F_i.

Proof: (conti)

choose ω to be zero, then $\sum F_{ci} \cdot v = 0$ for any choice of *v*, from which we conclude that in fact $\sum F_{ci} = \mathbf{0}$ is always true. This means that the constraint forces produce no net force

Similarly, choosing *v* to be 0 we see that $\sum -F_{ci} \cdot (r_i'^* \omega) = 0$ for any ω . Rewriting $F_{ci} \cdot (r_i'^* \omega)$ as $F_{ci}^{\,T} (r_i'^* \omega)$ we get that

$$\sum -F_{ci}^{\,T} r_i'^* \omega = \left(\sum -F_{ci}^{\,T} r_i'^* \right) \omega = 0$$

for any ω , so $\sum -F_{ci}^{\,T}\ r_i'^* = 0^{\mathrm{T}}$. Transposing, we have

$$\sum -(r_i'^*)^T F_{ci} = \sum (r_i')^* F_{ci} = \sum r_i' \times F_{ci} = \mathbf{0}$$

which means that the internal forces produce no net torque.

We can use the above to derive the rigid body equations of motion. The net force on each particle is the sum of the internal constraint force *F_{ci} and the external force F_i.*

Proof: (conti)

The acceleration $\ddot{r}_i$ of the *i-th* particle is

$$\ddot{r}_i = \frac{d}{dt}\dot{r}_i = \frac{d}{dt}(v - r_i'^* \omega) = \dot{v} - \dot{r}_i'^* \omega - r_i'^* \dot{\omega}.$$

Since each individual particle must obey Newton's law *f = ma,* or equivalently *ma − f = 0*, we have

$$m_i \ddot{r}_i - F_i - F_{ci} = m_i(\dot{v} - \dot{r}_i'^* \omega - r_i'^* \dot{\omega}) - F_i - F_{ci} = 0 \qquad (7\text{-}8)$$

for each particle.

To derive $\dot{P} = F = \sum F_i$, we sum the above equation over all the particles. We obtain

$$\sum m_i(\dot{v} - \dot{r}_i'^* \omega - r_i'^* \dot{\omega}) - F_i - F_{ci} = 0.$$

Breaking the large sum into smaller ones,

Proof: (conti)

$$\sum m_i(\dot{v} - \dot{r}_i'^*\omega - r_i'^*\dot{\omega}) - F_i - F_{ci} =$$

$$\sum m_i\dot{v} - \sum m_i\dot{r}_i'^*\omega - \sum m_i r_i'^*\dot{\omega} - \sum F_i - \sum F_{ci} =$$

$$\sum m_i\dot{v} - \left(\sum m_i\dot{r}_i'\right)^*\omega - \left(\sum m_i r_i'\right)^*\dot{\omega} - \sum F_i - \sum F_{ci} =$$

$$\sum m_i\dot{v} - \left(\frac{d}{dt}\sum m_i r_i'\right)^*\omega - \left(\sum m_i r_i'\right)^*\dot{\omega} - \sum F_i - \sum F_{ci} = \mathbf{0}.$$

Since we are in a center-of-mass coordinate system, eq. (7–6) from slide 20 tells us that $\sum m_i r_i' = \mathbf{0}$, which also means that $d(\sum m_i r_i')/dt = \mathbf{0}$. Removing terms with $\sum m_i r_i'$, and the term $\sum F_{ci}$ from the above equation yields

$$\sum m_i\dot{v} - \sum F_i = 0$$

or simply $M\dot{v} = \dot{P} = \sum F_i = F.$ Q.E.D.

Angular momentum

- most unintuitive concept! Nevertheless, makes equations simpler than using angular velocity
- constant angular momentum does not imply constant angular velocity
- Total angular momentum

$$\boxed{L(t) = I(t)\,\omega(t)}$$

where $I(t)$ is a 3x3 (rank two) matrix called *inertia tensor*

- The inertia tensor describes how the mass in a body is distributed relative to the body's center of mass
- $I(t)$ depends on the orientation of a body, but does not dependent on its translation
- Relationship between L(t) and total torque: $\boxed{\dot{L}(t) = \tau(t)}$

The inertia tensor

- scaling factor between angular momentum and angular velocity

$$I(t) = \sum \begin{pmatrix} m_i(r'^2_{iy} + r'^2_{iz}) & -m_i r'_{ix} r'_{iy} & -m_i r'_{ix} r'_{iz} \\ -m_i r'_{iy} r'_{ix} & m_i(r'^2_{ix} + r'^2_{iz}) & -m_i r'_{iy} r'_{iz} \\ -m_i r'_{iz} r'_{ix} & -m_i r'_{iz} r'_{iy} & m_i(r'^2_{ix} + r'^2_{iy}) \end{pmatrix}$$

where $r'_i = r_i(t) - x(t)$

- for an actual implementation, replace the finite sums with integrals over a body's volume
- however, computation should not be done in world space, but using body-space coordinates to compute the inertia tensor for any orientation *R(t)* in terms of a pre-computed integral in body-space coordinates (why and how?)
- The mass terms m_i are replaced by a density function

The inertia tensor

Note that

$$I(t) = \sum m_i r_i'^T r_i' \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} m_i r'^2_{ix} & m_i r'_{ix} r'_{iy} & m_i r'_{ix} r'_{iz} \\ m_i r'_{iy} r'_{ix} & m_i r'^2_{iy} & m_i r'_{iy} r'_{iz} \\ m_i r'_{iz} r'_{ix} & m_i r'_{iz} r'_{iy} & m_i r'^2_{iz} \end{pmatrix}$$

$$= \sum m_i ((r_i'^T r_i') E - r_i' r_i'^T)$$

where *E* is the 3x3 identity matrix.

Since $r_i' = R(t) r_{0_i}$ and $R(t) R(t)^T = E$, we have

The inertia tensor

$$
\begin{aligned}
I(t) &= \sum m_i((r_i'^T r_i')\mathbf{E} - r_i' r_i'^T) \\
&= \sum m_i((R(t)r_{0_i})^T(R(t)r_{0_i})\mathbf{E} - (R(t)r_{0_i})(R(t)r_{0_i})^T) \\
&= \sum m_i(r_{0_i}^T R(t)^T R(t) r_{0_i}\mathbf{E} - R(t)r_{0_i}r_{0_i}^T R(t)^T) \\
&= \sum m_i((r_{0_i}^T r_{0_i})\mathbf{E} - R(t)r_{0_i}r_{0_i}^T R(t)^T).
\end{aligned}
$$

Since $r_{0_i}^T r_{0_i}$ is a scalar, we can rearrange things by writing

$$
\begin{aligned}
I(t) &= \sum m_i((r_{0_i}^T r_{0_i})\mathbf{E} - R(t)r_{0_i}r_{0_i}^T R(t)^T) \\
&= \sum m_i(R(t)(r_{0_i}^T r_{0_i})R(t)^T\mathbf{E} - R(t)r_{0_i}r_{0_i}^T R(t)^T) \\
&= R(t)\left(\sum m_i((r_{0_i}^T r_{0_i})\mathbf{E} - r_{0_i}r_{0_i}^T)\right)R(t)^T.
\end{aligned}
$$

The inertia tensor

If we define I_{body} as the matrix

$$I_{body} = \sum m_i((r_{0_i}^T r_{0_i})E - r_{0_i} r_{0_i}^T)$$

then from the previous equation we have

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

Since I_{body} is specified in body-space, it is constant over the simulation. Thus, by pre-computing I_{body} for a body before the simulation begins, we can easily compute $I(t)$ from I_{body} and the orientation matrix *R(t)*.

Why is $\boxed{\dot{L}(t) = \tau(t)}$?

Proof: To obtain the above equation, we again start with equation (7-8). Multiplying both sides by $r_i'^*$ yields

$$r_i'^* m_i(\dot{v} - \dot{r}_i'^* \omega - r_i'^* \dot{\omega}) - r_i'^* F_i - r_i'^* F_{ci} = r_i'^* 0 = 0.$$

Summing over all the particles, we obtain

$$\sum r_i'^* m_i \dot{v} - \sum - \left(\sum m_i r_i'^* \dot{r}_i'^*\right) \omega - \left(\sum m_i r_i'^* r_i'^*\right) \dot{\omega} = \tau. \; r_i'^* F_i - \sum r_i'^* F_{ci} = 0.$$

Since $\sum r_i'^* F_{ci} = 0$ and $\sum m_i r_i' = 0$, we are left with

$$-\left(\sum m_i r_i'^* \dot{r}_i'^*\right) \omega - \left(\sum m_i r_i'^* r_i'^*\right) \dot{\omega} - \sum r_i'^* F_i = 0$$

or, recognizing that $\sum r_i'^* F_i = \sum r_i' \times F_i = \tau$,

$$-\left(\sum m_i r_i'^* \dot{r}_i'^*\right) \omega - \left(\sum m_i r_i'^* r_i'^*\right) \dot{\omega} = \tau. \qquad (7\text{-}9)$$

Proof: (conti.)

It is easy to verify that the matrix $-a^*a^*$ is equivalent to the matrix $(a^T a)E - a\,a^T$ where E is the 3x3 identity matrix. Thus

$$\sum -m_i r_i'^* r_i'^* = \sum m_i((r_i'^T r_i')E - r_i' r_i'^T) = I(t).$$

Substituting into equation (7–9), this yields

$$\left(\sum -m_i r_i'^* \dot{r}_i'^*\right)\omega + I(t)\dot{\omega} = \tau. \qquad (7\text{-}10)$$

Since $\dot{r}_i' = \omega \times r_i'$ and $\dot{r}_i'^*\omega = -\omega \times r_i'$, we can write

$$\begin{aligned}\sum m_i \dot{r}_i'^* r_i'^* \omega &= \sum m_i(\omega \times r_i')^*(-\omega \times r_i') \\ &= \sum -m_i(\omega \times r_i') \times (\omega \times r_i') = 0.\end{aligned}$$

Proof: (conti.)

Thus, we can add $-\sum m_i \dot{r}_i'^* r_i'^* = 0$ to equation (7-10) to obtain

$$\left(\sum -m_i r_i'^* \dot{r}_i'^* - m_i \dot{r}_i'^* r_i'^*\right)\omega + I(t)\dot{\omega} = \tau.$$

Finally, since

$$\dot{I}(t) = \frac{d}{dt}\sum -m_i r_i'^* r_i'^* = \sum -m_i r_i'^* \dot{r}_i'^* - m_i \dot{r}_i'^* r_i'^*$$

we have

$$\dot{I}(t)\omega + I(t)\dot{\omega} = \frac{d}{dt}(I(t)\omega) = \tau.$$

Since $L(t) = I(t)\omega(t)$, this leaves us with the result that

$$\dot{L}(t) = \tau.$$

Rigid Body Equations of Motion

- ready to define the state vector *X(t)*

$$X(t) = (\underbrace{x(t), R(t)}_{\text{spatial information}}, \underbrace{P(t), L(t)}_{\text{velocity information}})^T$$

- body mass *M* and body space inertia tensor I_{body} are constants known before the simulation begins

- auxiliary quantities $I(t), \omega(t)$ and $v(t)$ are computes by

$$v(t) = \frac{P(t)}{M}, \quad I(t) = R(t) I_{body} R(t)^T \quad \text{and} \quad \omega(t) = I(t)^{-1} L(t)$$

- derivative *d**X(t)**/dt* is

$$\frac{d}{dt}\mathbf{X}(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \omega(t)^* R(t) \\ F(t) \\ \tau(t) \end{pmatrix}$$

Computing the derivative of *X(t)*

- consider an implementation of the function *dxdt()* for rigid bodies

- representing a rigid body by the structure

```
struct RigidBody {
    /* Constant quantities */
    double  mass;           /* mass M */
    matrix  Ibody,          /* I_body */
            Ibodyinv;       /* I_body^-1 (inverse of I_body) */

    /* State variables */
    triple  x;              /* x(t) */
    matrix  R;              /* R(t) */
    triple  P,              /* P(t) */
            L;              /* L(t) */
```


(Assume the datatypes matrix and triple are available)

Computing the derivative of *X(t)* (conti.)

```
    /* Derived quantities (auxiliary variables) */
    matrix  Iinv;              /* I^{-1}(t) */
    triple  v,                 /* v(t) */
            omega;             /* ω(t) */

    /* Computed quantities */
    triple  force,             /* F(t) */
            torque;            /* τ(t) */
};
```

- assume a global array of bodies

```
RigidBody Bodies[NBODIES];
```

- constants *mass*, *Ibody* and *Ibodyinv* are calculated for each member of Bodies, before simulation begins
- initial values are assigned to the state variables x, R, P and L of each member of Bodies

Computing the derivative of *X(t)* (conti.)

- communicate with the differential equation solver *ode* by passing arrays of real numbers. Several bookkeeping routines are required:

```
/* Copy the state information into an array */
void StateToArray(RigidBody *rb, double *y)
{
    *y++ = rb->x[0];            /* x component of position */
    *y++ = rb->x[1];            /* etc. */
    *y++ = rb->x[2];

    for(int i = 0; i < 3; i++)  /* copy rotation matrix */
        for(int j = 0; j < 3; j++)
            *y++ = rb->R[i,j];

    *y++ = rb->P[0];
    *y++ = rb->P[1];
    *y++ = rb->P[2];
```

Computing the derivative of *X(t)* (conti.)

```
    *y++ = rb->L[0];
    *y++ = rb->L[1];
    *y++ = rb->L[2];
}
```

and

```
/* Copy information from an array into the state variables */
void ArrayToState(RigidBody *rb, double *y)
{
    rb->x[0] = *y++;
    rb->x[1] = *y++;
    rb->x[2] = *y++;

    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            rb->R[i,j] = *y++;
```

Computing the derivative of *X(t)* (conti.)

```
    rb->P[0] = *y++;
    rb->P[1] = *y++;
    rb->P[2] = *y++;

    rb->L[0] = *y++;
    rb->L[1] = *y++;
    rb->L[2] = *y++;

    /* Compute auxiliary variables... */

    /* v(t) = P(t)/M */
    rb->v = rb->P / mass;

    /* I^-1(t) = R(t) I_body^-1 R(t)^T */
    rb->Iinv = R * Ibodyinv * Transpose(R);

    /* ω(t) = I^-1(t) L(t) */
    rb->omega = rb->Iinv * rb->L;
}
```

Computing the derivative of *X(t)* (conti.)

- Transfers between all the members of Bodies and an array y of size 18 x NBODIES are implemented as

```
#define STATE_SIZE      18

void ArrayToBodies(double x[])
{
    for(int i = 0; i < NBODIES; i++)
        ArrayToState(&Bodies[i], &x[i * STATE_SIZE]);
}
```

and

```
void BodiesToArray(double x[])
{
    for(int i = 0; i < NBODIES; i++)
        StateToArray(&Bodies[i], &x[i * STATE_SIZE]);
}
```

Computing the derivative of *X(t)* (conti.)

- the following routine computes force *F(t)* and torque $\omega(t)$:

```
void    ComputeForceAndTorque(double t, RigidBody *rb);
```

- *dxdt( )* can be defined as follows:

```
void dxdt(double t, double x[], double xdot[])
{
    /* put data in x[] into Bodies[] */
    ArrayToBodies(x);

    for(int i = 0; i < NBODIES; i++)
    {
        ComputeForceAndTorque(t, &Bodies[i]);
        DdtStateToArray(&Bodies[i],
                        &xdot[i * STATE_SIZE]);
    }
}
```

Computing the derivative of *X(t)* (conti.)

- the following routine computes force *F(t)* and torque $\omega(t)$:

```
void    ComputeForceAndTorque(double t, RigidBody *rb);
```

- *dxdt( )* can be defined as follows:

```
void dxdt(double t, double x[], double xdot[])
{
    /* put data in x[] into Bodies[] */
    ArrayToBodies(x);

    for(int i = 0; i < NBODIES; i++)
    {
        ComputeForceAndTorque(t, &Bodies[i]);
        ddtStateToArray(&Bodies[i],
                        &xdot[i * STATE_SIZE]);
    }
}
```

does the real work!

Computing the derivative of *X(t)* (conti.)

```
void ddtStateToArray(RigidBody *rb, double *xdot)
{
    /* copy d/dt x(t) = v(t) into xdot */
    *xdot++ = rb->v[0];
    *xdot++ = rb->v[1];
    *xdot++ = rb->v[2];

    /* Compute Ṙ(t) = ω(t)* R(t) */
    matrix  Rdot = Star(rb->omega) * rb->R;

    /* copy Ṙ(t) into array */
    for(int i = 0; i < 3; i++)
        for(int j = 0; j < 3; j++)
            *xdot++ = Rdot[i,j];

    *xdot++ = rb->force[0];      /* d/dt P(t) = F(t) */
    *xdot++ = rb->force[1];
    *xdot++ = rb->force[2];
```

Computing the derivative of *X(t)* (conti.)

```
    *xdot++ = rb->torque[0];        /* d/dt L(t) = τ(t) */
    *xdot++ = rb->torque[1];
    *xdot++ = rb->torque[2];
}
```

- The routine Star, used to calculate $\dot{R}(t)$ is defined as

```
    matrix   Star(triple a);
```

and returns the matrix

$$\begin{pmatrix} 0 & -\mathtt{a[2]} & \mathtt{a[1]} \\ \mathtt{a[2]} & 0 & -\mathtt{a[0]} \\ -\mathtt{a[1]} & \mathtt{a[0]} & 0 \end{pmatrix}$$

See slide 14 for definition of *a**

Computing the derivative of *X(t)* (conti.)

- performing a simulation for 10 seconds, calling *Display-Bodies* every (1/24)-th of a second to display the bodies :

```
void RunSimulation()
{
    double  x0[STATE_SIZE * NBODIES],
            xFinal[STATE_SIZE * NBODIES];

    InitStates();
    BodiesToArray(xFinal);

    for(double t = 0; t < 10.0; t += 1./24.)
    {
        /* copy xFinal back to x0 */
        for(int i = 0; i < STATE_SIZE * NBODIES; i++)
        {
            x0[i] = xFinal[i];
```

initialize the state variables of all NBODIES of rigid bodies

Computing the derivative of *X(t)* (conti.)

```
for(double t = 0; t < 10.0; t += 1./24.)
{
    /* copy xFinal back to x0 */
    for(int i = 0; i < STATE_SIZE * NBODIES; i++)
    {
        x0[i] = xFinal[i];
    ode(x0, xFinal, STATE_SIZE * NBODIES,
        t, t+1./24., dxdt);

    /* copy d/dt X(t + 1/24) into state variables */

    ArrayToBodies(xFinal);
    DisplayBodies();
}
}
```

End of Physically Based Animation II