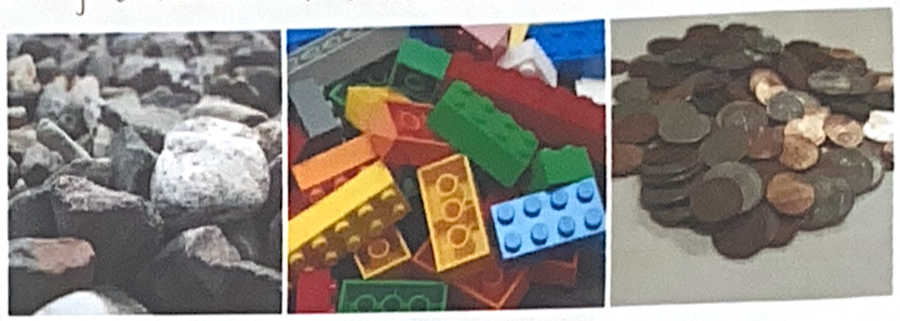
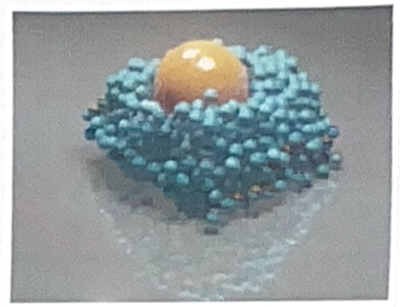
- Rigid Body

Our living enviroment is stuffed with rigid objects.



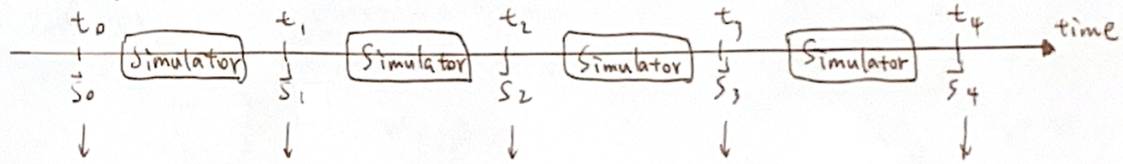
In virtual worlds, we want to simulate rigid body as well.



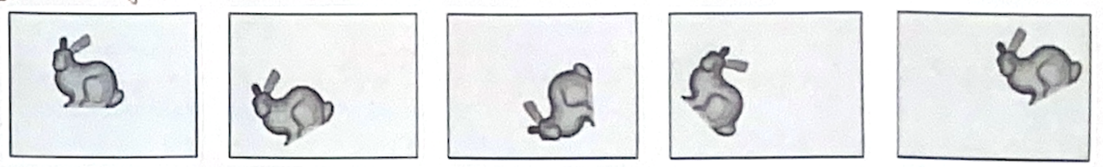
※ Rigid Body Simulation:

The goal of simulation is to update the state variable $\vec{S}_n$ over time.

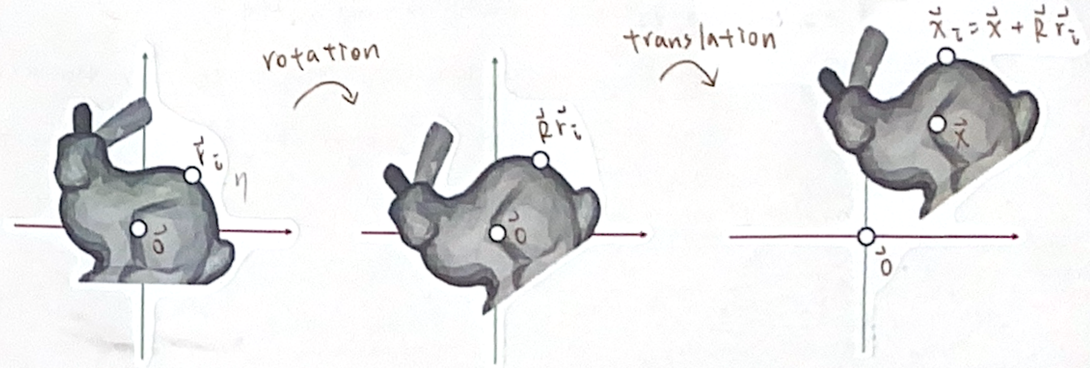
Physics Engine



Game Engine



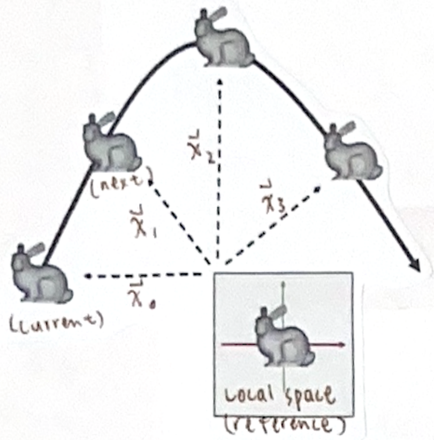
* If rigid body cannot deform, its motions consists of two parts:



Local Space (Reference)

World Space (current)

※ Translational Motion:



For translational motion, the state variable contains the position $\vec{x}$ and the velocity $\vec{v}$

* Equation of motion:

$$\begin{cases} \vec{v} = \vec{a} = \frac{\vec{F}}{m} \\ \dot{\vec{x}} = \vec{v} \end{cases}$$

user-specified variable

Euler-Cromer:

$$\vec{v}_{n+1} = \vec{v}_n + \Delta t M^{-1} \vec{f}_n \quad \text{— Explicit}$$

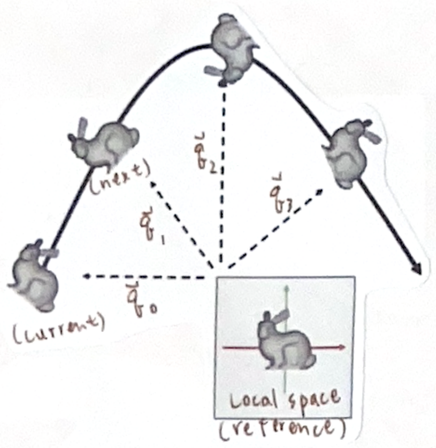
$$\vec{x}_{n+1} = \vec{x}_n + \Delta t \vec{v}_{n+1} \quad \text{— Implicit}$$

* Type of force:

1. Gravity force: $\vec{f}_g = M\vec{g}$
2. Drag force: $\vec{f}_{drag} = -\sigma \vec{v}_n$ ↳ drag coefficient

reduce $\vec{v}$ → $\vec{v}_{n+1} = \alpha \vec{v}_n$ ↳ decay coefficient

※ Rotational Motion:



The rotation from reference to current, choose quaternion $\vec{q}$ to represent the orientation.

$$\begin{cases} \dot{w} = \alpha = \frac{\vec{\tau}}{I} & \text{torque} \\ & \text{inertia} \\ \dot{q} = w \end{cases}$$

$$\rightarrow \begin{cases} \vec{w}_{n+1} = \vec{w}_n + \Delta t (\vec{I}_n)^{-1} \vec{\tau}_n \\ \vec{q}_{n+1} = \vec{q}_n + \left[0 \quad \frac{\Delta t}{2} w_{n+1}\right] \times q_n \end{cases}$$

▲ Use a 3D vector $\vec{w}$ to denote angular velocity.

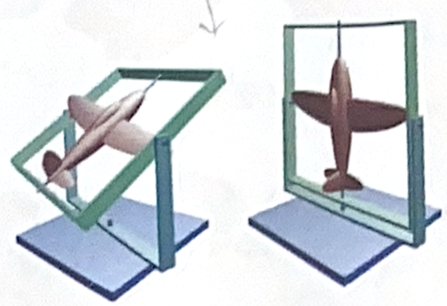
{ the direction of $\vec{w}$ is the axis
the magnitude of $\vec{w}$ is the speed

* Rotation Represented:

1. Eular Angles: ex. (r_x, r_y, r_z)

Pro: intuitive! each axial rotation uses an angle.

Con: hard to define time derivative
lose DoFs in certain statuses: gimble lock

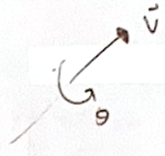


2. Matrix: ex. $\vec{R} = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}$

Pro: friendly apply rotation to each vertex

Con: too redundancy: 9 elements for 3 DoFs
non-intuitive
hard to define time derivative

3. quaternion: ex. $\vec{q} = [s \quad \vec{v}]$ ↳ 3D vertex part, accounting for $\vec{i}\vec{j}\vec{k}$
↳ scaler part



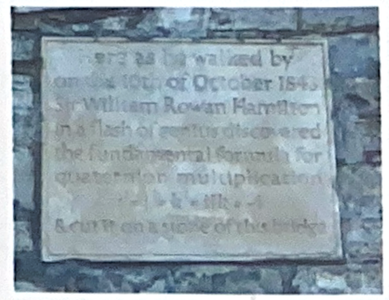
$$\begin{cases} \vec{q} = [\cos\frac{\theta}{2} \quad \vec{v}] \\ \|\vec{q}\| = 1 \end{cases} \rightarrow \begin{cases} \vec{q} = [\cos\frac{\theta}{2} \quad \vec{v}] \\ \|\vec{v}\| = \sin\frac{\theta}{2} \end{cases}$$

▲ convert to matrix:

$$\vec{R} = \begin{bmatrix} s^2+x^2-y^2-z^2 & 2(xy-sz) & 2(xz+sy) \\ 2(xy+sz) & s^2-x^2+y^2-z^2 & 2(yz-sx) \\ 2(xz-sy) & 2(yz+sx) & s^2-x^2-y^2+z^2 \end{bmatrix}$$

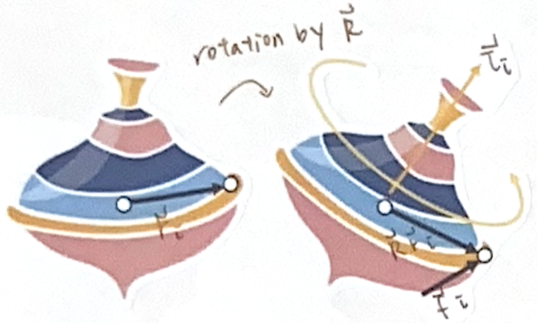
▲ quaternion arithematic: more 可以看 GAMES105 02

- scalar-quaternion multiplication: $a\vec{q} = [as \quad a\vec{v}]$
- addition/subtraction: $\vec{q}_1 \pm \vec{q}_2 = [s_1 \pm s_2 \quad \vec{v}_1 \pm \vec{v}_2]$
- multiplication: $\vec{q}_1 \times \vec{q}_2 = [s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2 \quad s_1\vec{v}_2 + s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2]$
- magnitude: $\|\vec{q}\| = \sqrt{s^2 + \vec{v} \cdot \vec{v}}$
- $i^2 = j^2 = k^2 = ijk = -1$
  $ij = k, \quad ji = -k$
  $jk = i, \quad kj = -i$
  $ki = j, \quad ik = -j$



★ Torque:

A torque is the rotational equivalent of a force. It describes the rotational tendency caused by a force.



$$\vec{\tau}_i = (\vec{R}\vec{r}_i) \times \vec{f}_i$$

$$\rightarrow \vec{\tau} = \sum_i \tau_i$$

a. $\vec{\tau}_i \perp \vec{R}\vec{r}_i$, $\vec{\tau}_i \perp \vec{f}_i$

b. $\vec{\tau}_i \propto \|\vec{R}\vec{r}_i\|$, $\vec{\tau}_i \propto \|\vec{f}_i\|$

$\vec{\tau}_i \propto \sin\theta$

★ Inertia Tensor:

Similar to mass, an inertia tensor describes the resistance to rational tendency caused by torque. But different from mass, it's not a constant. It's a matrix! The mass inverse is the resistance.

$$\vec{I}_{ref} = \sum m_i (\vec{r}_i^T \vec{r}_i \vec{1} - \vec{r}_i \vec{r}_i^T)$$

$$\vec{I} = \sum m_i (\vec{r}_i^T \vec{R}^T \vec{R} \vec{r}_i \vec{1} - \vec{R}\vec{r}_i \vec{r}_i^T \vec{R}^T)$$

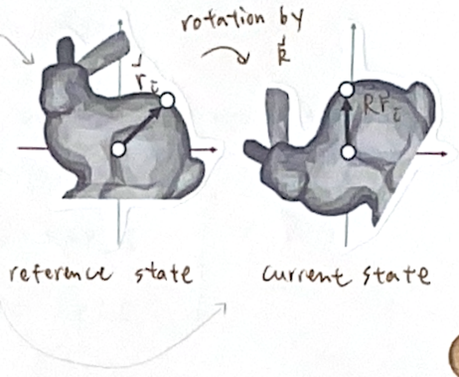
物件旋轉後的慣性 tensor 會改變

$$= \sum m_i (\vec{R}\vec{r}_i^T \vec{r}_i \vec{1} \vec{R}^T - \vec{R}\vec{r}_i \vec{r}_i^T \vec{R}^T)$$

$$= \sum m_i \vec{R}(\vec{r}_i^T \vec{r}_i \vec{1} - \vec{r}_i \vec{r}_i^T)\vec{R}^T$$

$$= \vec{R}\vec{I}_{ref}\vec{R}^T$$

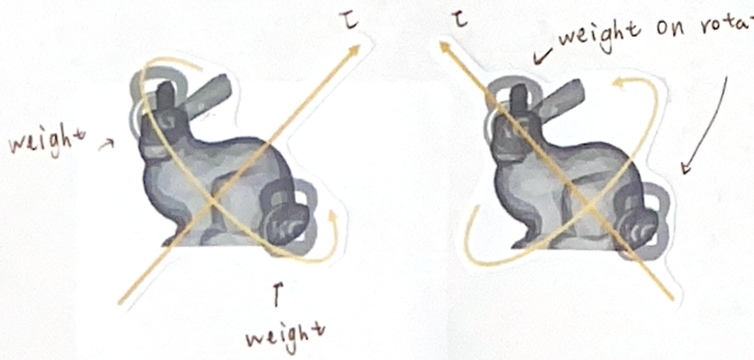
→ 但如果在先算 $\vec{I}_{ref}$，使迭代過程在計算量更改!



• example:

Same torque τ, the rotational motion on the right hand side would be stronger than the left hand side. Because it has smaller inertia.

faster



Conclusion: the inertia is related to torque.

RB3

★ Short summary:

▲ State & physical quantities

Translational		Rotational	
Mass M	Force $\vec{f}$	Inertia $\vec{I}$	Torque $\vec{\tau}$
Velocity $\vec{v}$	Position $\vec{x}$	Angular velocity $\vec{w}$	Quaternion $\vec{q}_r$

▲ Simulator loop: n表時間步，i表頂點

Translational

$$\vec{f}_{n,i} = \text{Force}(\vec{x}_{n,i}, \vec{v}_{n,i})$$

$$\vec{f}_n = \sum_i \vec{f}_{n,i}$$

$$\vec{v}_n, \vec{x}_n \rightarrow \vec{v}_{n+1}, \vec{x}_{n+1}$$

$$\vec{v}_{n+1} = \vec{v}_n + \Delta t\, M^{-1} \vec{f}_n$$

$$\vec{x}_{n+1} = \vec{x}_n + \Delta t\, \vec{v}_{n+1}$$

Rotational

$$\vec{R}_n = \text{Matrix.rotation}(\vec{q}_{r,n})$$

$$\vec{\tau}_{n,i} = (\vec{R}_n \vec{r}_i) \times \vec{f}_{n,i}$$

$$\vec{\tau}_n = \sum_i \vec{\tau}_{n,i}$$

$$\vec{w}_n, \vec{q}_n \rightarrow \vec{w}_{n+1}, \vec{q}_{n+1}$$

$$\vec{I}_n = \vec{R}_n \vec{I}_{ref} \vec{R}_n^T$$

$$\vec{w}_{n+1} = \vec{w}_n + \Delta t\, \vec{I}_n^{-1} \vec{\tau}_n$$

$$\vec{q}_{r,n+1} = \vec{q}_{r,n} + \left[0 \quad \frac{\Delta t}{2} \vec{w}_{n+1}\right] \times \vec{q}_{r,n}$$

▲ initial condition:

Δt : time step

M : mass

$$\vec{I}_{ref} = \sum m_i (\vec{r}_i^T \vec{r}_i \vec{1} - \vec{r}_i \vec{r}_i^T)$$

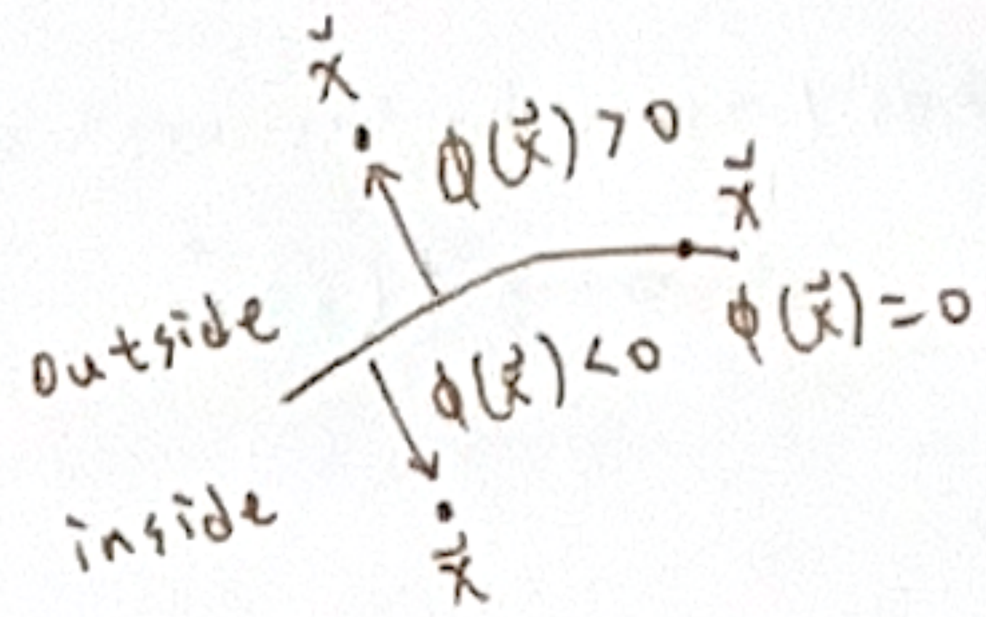
★ Appendix: Quaternion Derivation   Pixar appendix B.

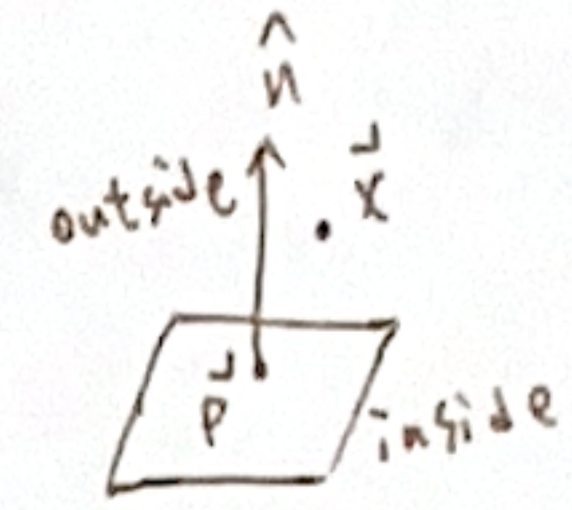
-ẋ- Collision Detection

- Particle Collision Detection:

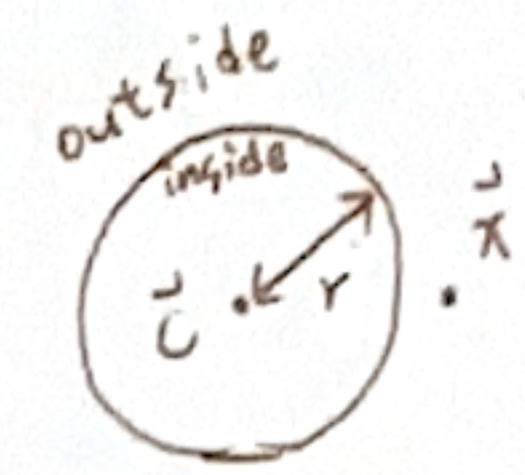
  Use signed distance function $\phi(\vec{x})$ defines the distance from $\vec{x}$ to a surface with sign. The sign indicates on which side $\vec{x}$ is located.

- example:

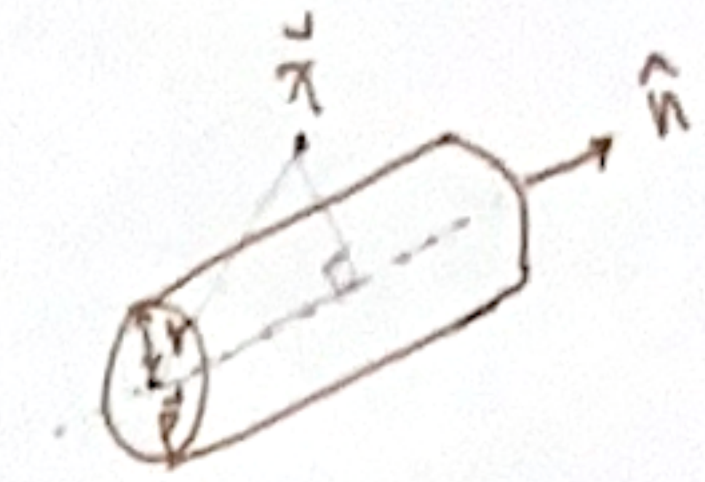




$$\phi(\vec{x}) = (\vec{x} - \vec{p}) \cdot \hat{n}$$

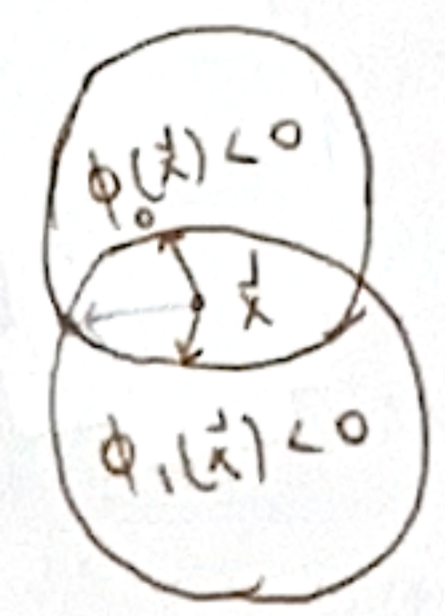


$$\phi(\vec{x}) = \|\vec{x} - \vec{c}\| - r$$



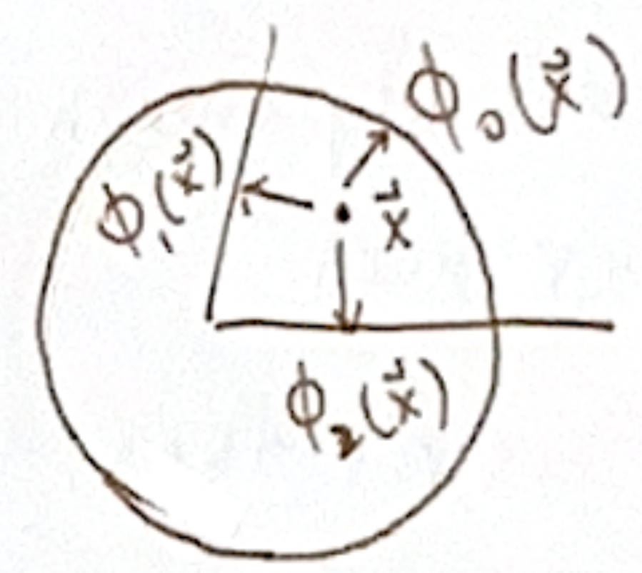
$$\phi(\vec{x}) = \sqrt{\|\vec{x} - \vec{p}\|^2 - ((\vec{x} - \vec{p}) \cdot \hat{n})^2} - r$$

- union of signed distance function:



if $\phi_0(\vec{x}) < 0$ or $\phi_1(\vec{x}) < 0$
    then inside
    $\phi(\vec{x}) \approx \min(\phi_0(\vec{x}), \phi_1(\vec{x}))$
else outside → correct near outer boundary
    $\phi(\vec{x}) = \min(\phi_0(\vec{x}), \phi_1(\vec{x}))$

- intersection of signed distance function:



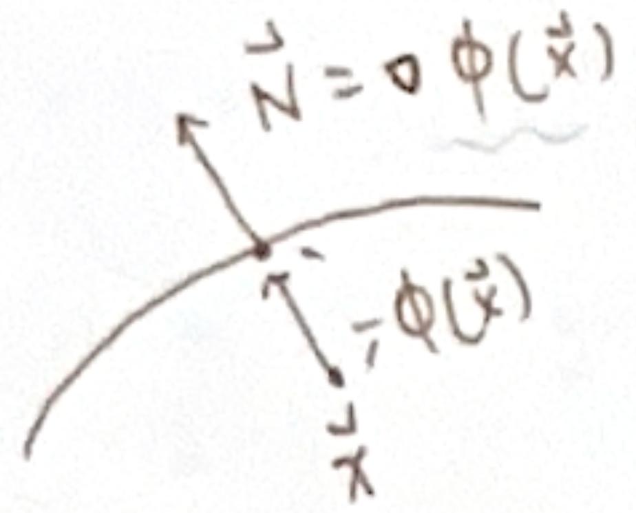
if $\phi_0(\vec{x}) < 0$ and $\phi_1(\vec{x}) < 0$ and $\phi_2(\vec{x}) < 0$
    then inside → φ是負的，因此 max 表最近的那個
    $\phi(\vec{x}) = \max(\phi_0(\vec{x}), \phi_1(\vec{x}), \phi_2(\vec{x}))$
else outside

- Particle Collision Responce:

1. Quadratic Penalty Method:

A penalty method applied a penalty force in the next update.

When the penalty potential is quadratic, the force is linear.



使φ(x)最快變大的方向，也就是法向量

$\phi(\vec{x}) < 0$ —no→ Done

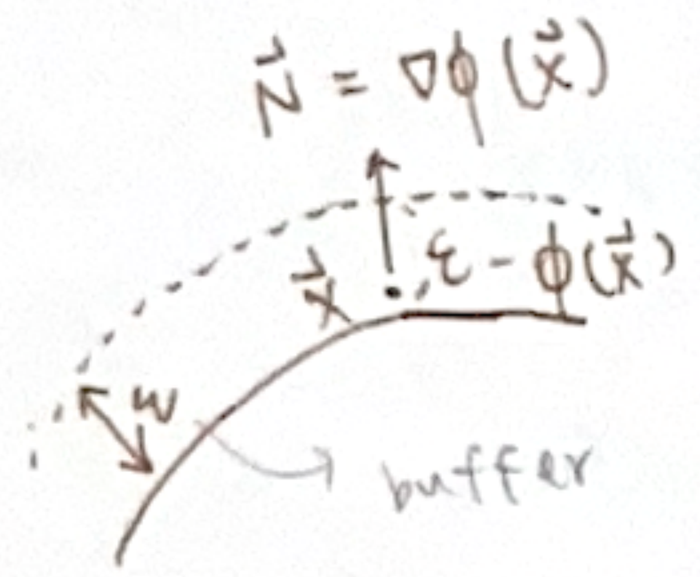
↓ yes

Apply force in the next update

$\vec{f} = -k\,\phi(\vec{x})\,\vec{N}$ (k: penalty strength)

▲ Problem: there will have penetration artifacts in the animation.

→ Solve: a buffer helps lessen the penetration issue. But it cannot strictly prevent penetration, no matter how large k is.



※ 只要 x 掉到 ε 內的範圍內，就給一個力。

$\phi(\vec{x}) < \varepsilon$ —no→ Done

↓ yes

Apply force in the next update

$\vec{f} = k(\varepsilon - \phi(\vec{x}))\,\vec{N}$

▲ Problem: if k is too small, the force is not big enough to push out $\vec{x}$. if k is too big, $\vec{x}$ will overshoot.

→ solve: a log-barrier penalty potential ensures that the force can be large enough. But assume $\phi(\vec{x}) < 0$ NEVER happen! To achieve that, it needs to adjust Δt.

Always apply the penalty force

$\vec{f} = \rho \frac{1}{\phi(\vec{x})} \vec{N}$ 距離越近力越大 (ρ: 衰減速度)

▲ Problem: 1. still, overshooting

2. frictional contacts are difficult to handle

* must adjust time step

2. Impulse method:

An impulse method assumes that collision changes the position and the velocity all of SUDDEN. 即刻性的改變!!

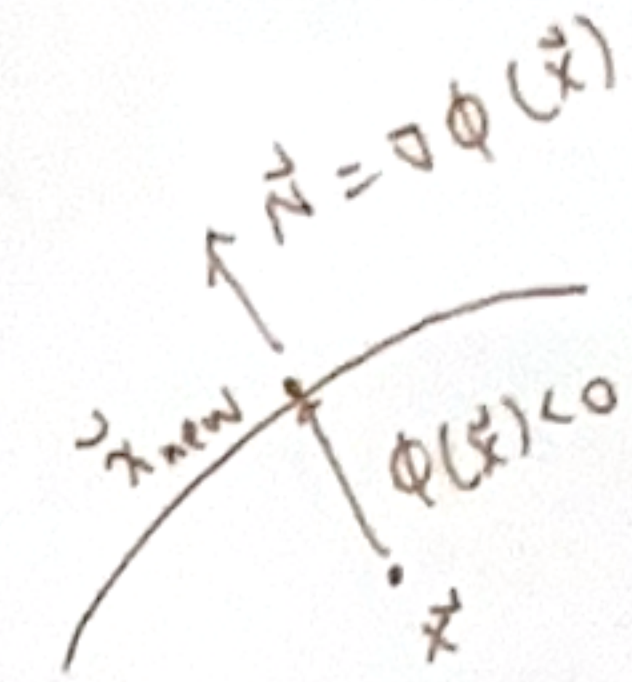
▲ update position:

$\phi(\vec{x}) < 0$ —no→ Done

↓ yes

Collision: $\vec{x}_{new} = \vec{x} + |\phi(\vec{x})|\,\vec{N}$ (長度, 方向)

$= \vec{x} - \phi(\vec{x})\,\nabla\phi(\vec{x})$



▲ update velocity:

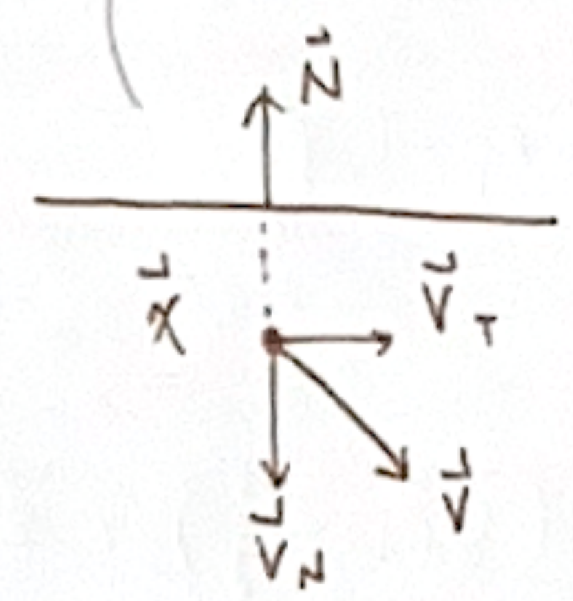
$\vec{v} \cdot \vec{N} < 0$ —no→ Done

↓ yes

origin $\vec{v}$: $\vec{v}_N = (\vec{v} \cdot \vec{N})\,\vec{N}$, $\vec{v}_T = \vec{v} - \vec{v}_N$ → $\vec{v}_N^{new} = -\mu_N \vec{v}_N$, $\vec{v}_T^{new} = a\,\vec{v}_T$ → $\vec{v}^{new} = \vec{v}_N^{new} + \vec{v}_T^{new}$

負號表示反彈

μ_N 為彈性係數，且 $\vec{v}_N^{new} < \vec{v}_N$，因此 $\mu_N \in [0,1]$



a 為摩擦造成切方向的衰減

a should be minimized but not violating Coulomb's law: (摩擦會造成速度下降max)

$\|\vec{v}_T^{new} - \vec{v}_T\| \le \mu_T \|\vec{v}_N^{new} - \vec{v}_N\|$

→ $(1-a)\|\vec{v}_T\| \le \mu_T(1+\mu_N)\|\vec{v}_N\|$

→ $a = \max\left(1 - \frac{\mu_T(1+\mu_N)\|\vec{v}_N\|}{\|\vec{v}_T\|},\ 0\right)$

(first term: dynamic fiction; 0: static fiction)

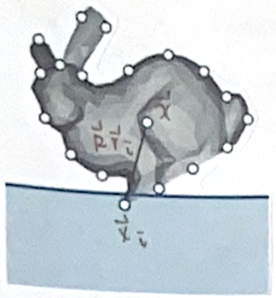
static fiction: 摩擦力最多使速度歸零，無法反轉速度方向

★ Short summary: (ex. 摩擦來自物體表面不平整，但這樣時沒有 物理世界本質!)

Penalty method: 好實作，物件會穿透，無法完全停下來，ex 衣服、彈性體多

Impulse method: 可精確控制 friction 和反彈的位置 ex Rigid body 多

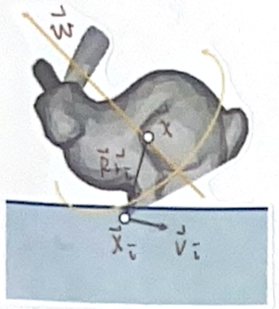
- Rigid Body Detection:



When the body is made of many vertices, we can detect its collision by testing each vertex:

$$\vec{x}_i = \vec{x} + \vec{R}\vec{r}_i$$

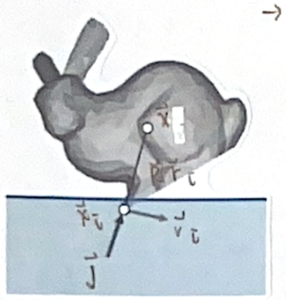
- Rigid Body Response by Impulse:



For vertex i:
$$\begin{cases} \vec{x}_i = \vec{x} + \vec{R}\vec{r}_i \\ \vec{v}_i = \vec{v} + \vec{w} \times \vec{R}\vec{r}_i \end{cases}$$
($\vec{v}$: linear velocity, $\vec{w}$: angular velocity)

Problem: We can't directly modify $\vec{x}_i$ or $\vec{v}_i$, since they are not state variables.

→ Solve: use an impulse $\vec{J}$ to modify $\vec{v}$ and $\vec{w}$ (of M的)



$$\begin{cases} \vec{v}^{new} = \vec{v} + \frac{1}{M}\vec{J} \\ \vec{w}^{new} = \vec{w} + \vec{I}^{-1}(\vec{R}\vec{r}_i \times \vec{J}) \end{cases}$$

Therefore, new velocity at vertex i:

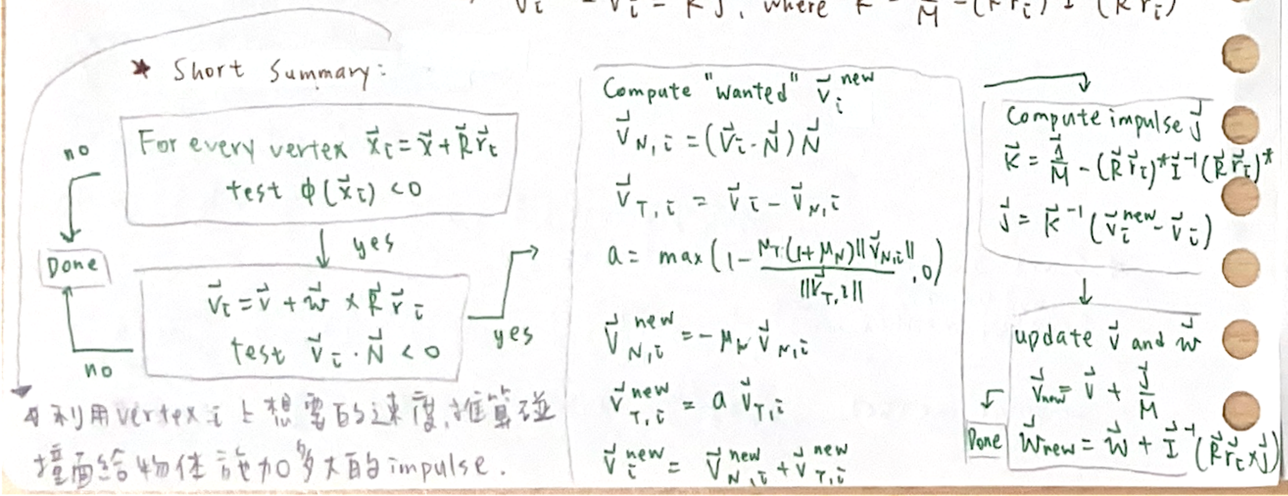
$$\vec{v}_i^{new} = \vec{v}^{new} + \vec{w}^{new} \times \vec{R}\vec{r}_i$$
$$= \vec{v} + \frac{1}{M}\vec{J} + (\vec{w} + \vec{I}^{-1}(\vec{R}\vec{r}_i \times \vec{J})) \times \vec{R}\vec{r}_i$$

Cross product as a matrix product: $\vec{r} \times \vec{q} = \begin{bmatrix} r_y q_z - r_z q_y \\ r_z q_x - r_x q_z \\ r_x q_y - r_y q_x \end{bmatrix} = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} = \vec{r}^* \vec{q}$

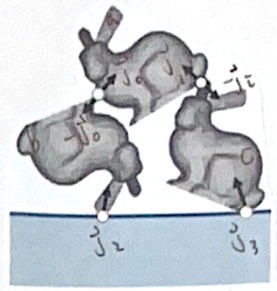
$$= \vec{v}_i + \frac{1}{M}\vec{J} - \vec{R}\vec{r}_i \times (\vec{I}^{-1}(\vec{R}\vec{r}_i \times \vec{J}))$$
$$\rightarrow = \vec{v}_i + \frac{1}{M}\vec{J} - (\vec{R}\vec{r}_i)^* \vec{I}^{-1}(\vec{R}\vec{r}_i)^* \vec{J}$$

$$\rightarrow \vec{v}_i^{new} - \vec{v}_i = \vec{K}\vec{J}, \text{ where } \vec{K} = \frac{1}{M} - (\vec{R}\vec{r}_i)^* \vec{I}^{-1}(\vec{R}\vec{r}_i)^*$$

★ Short Summary:



▲ Multi-object Rigid Body Collision:



Relative velocity at joints:

$$\begin{cases} \vec{v}_0^{new} - \vec{v}_0 = \vec{K}_{a00}\vec{J}_0 + \vec{K}_{a01}\vec{J}_1 - (-\vec{K}_{b00}\vec{J}_0 + \vec{K}_{b02}\vec{J}_2) \\ \vec{v}_1^{new} - \vec{v}_1 = \vec{K}_{a10}\vec{J}_0 + \vec{K}_{a11}\vec{J}_1 - (-\vec{K}_{c11}\vec{J}_0 + \vec{K}_{c13}\vec{J}_3) \\ \vec{v}_2^{new} - \vec{v}_2 = -\vec{K}_{b20}\vec{J}_0 + \vec{K}_{b22}\vec{J}_2 \\ \vec{v}_3^{new} - \vec{v}_3 = -\vec{K}_{c31}\vec{J}_1 + \vec{K}_{c33}\vec{J}_3 \end{cases}$$

$$\rightarrow \begin{bmatrix} \vec{K}_{a00}+\vec{K}_{b00} & \vec{K}_{a01} & -\vec{K}_{b02} & 0 \\ \vec{K}_{a10} & \vec{K}_{a11}+\vec{K}_{c11} & 0 & -\vec{K}_{c13} \\ -\vec{K}_{b20} & 0 & \vec{K}_{b22} & 0 \\ 0 & -\vec{K}_{c31} & 0 & \vec{K}_{c33} \end{bmatrix} \begin{bmatrix} \vec{J}_0 \\ \vec{J}_1 \\ \vec{J}_2 \\ \vec{J}_3 \end{bmatrix} = \begin{bmatrix} \Delta\vec{v}_0 \\ \Delta\vec{v}_1 \\ \Delta\vec{v}_2 \\ \Delta\vec{v}_3 \end{bmatrix}$$

where $\vec{K}_{a01}\vec{J}_1$ stands for the velocity change of bunny a at joint 0, caused by impulse $\vec{J}_1$.

- Shape Matching: ★ no physics, an optimization problem

Allow each vertex to have its own velocity, so it can move itself.

Step.1 move vertecies independently by its velocity, with collision and friction being handled.

Step.2 enforce the rigidity constraint to become a rigid body again.

unknown $\vec{c}$ & $\vec{R}$: $\{\vec{c}, \vec{R}\} = \text{argmin} \sum_i \frac{1}{2}\|\vec{c} + \vec{R}\vec{r}_i - \vec{y}_i\|^2$

$\rightarrow \{\vec{c}, \vec{A}\} = \text{argmin} \sum_i \frac{1}{2}\|\vec{c} + \vec{A}\vec{r}_i - \vec{y}_i\|^2$ ($\vec{A}$: any matrix; objective 方)

$\frac{\partial E}{\partial c}$ $\frac{\partial E}{\partial A}$

$\frac{\partial E}{\partial c} = \sum_i \vec{c} + \vec{A}\vec{r}_i - \vec{y}_i = \sum_i \vec{c} - \vec{y}_i = 0$

$\rightarrow \vec{c} = \frac{1}{N}\sum_i \vec{y}_i$ → CofM位置不變

$\frac{\partial E}{\partial A} = \sum_i (\vec{c} + \vec{A}\vec{r}_i - \vec{y}_i)\vec{r}_i^T = 0$

$\rightarrow \vec{A} = \left(\sum_i (\vec{y}_i - \vec{c})\vec{r}_i^T\right)\left(\sum_i \vec{r}_i \vec{r}_i^T\right)^{-1} = \vec{R}\vec{S}$

Appendix: Polar decomposition

★ Short Summary :

Independent update for every vertex

$$\vec{f}_{tmp,i} = Force(\vec{v}_{n,i}, \vec{x}_{n,i})$$

$$\vec{v}_{tmp,i} = \vec{v}_{n,i} + \Delta t m_i^{-1} \vec{f}_{tmp,i}$$

$$\vec{y}_{tmp,i} = \vec{x}_{n,i} + \Delta t \vec{v}_{n,i}$$

→

Rigidify the vertices

$$\vec{c} = \frac{1}{N} \sum_i \vec{y}_{tmp,i}$$

$$\vec{A} = \left(\sum_i (\vec{y}_{tmp,i} - \vec{c}) \vec{r}_i^T\right) \left(\sum_i \vec{r}_i \vec{r}_i^T\right)^{-1}$$

$$\vec{R} = Polar(\vec{A})$$

↓

Update $\vec{v}_{n+1,i}$ and $\vec{x}_{n+1,i}$

$$\vec{v}_{n+1,i} = (\vec{c} + \vec{R}\vec{r}_i - \vec{x}_{n,i})/\Delta t$$

$$\vec{x}_{n+1,i} = \vec{c} + \vec{R}\vec{r}_i$$

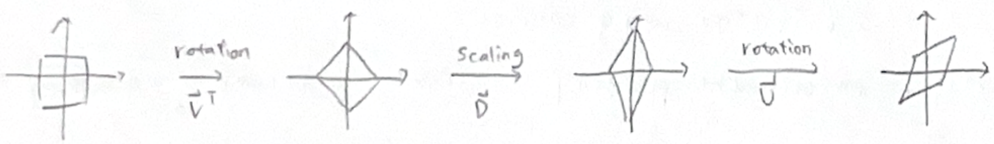
←

Done

1. easier to implement and compatible with other nodal systems, i.e., cloth, soft bodies and even particle fluids.
2. difficult to strictly enforce friction and constrains.

★ Appendix: Polar Decomposition

Singular value decomposition says that any matrix can be decomposed into : rotation, scaling, and rotation: $\vec{A} = \vec{U}\vec{D}\vec{V}^T$



However, we can rotate the object back before the final rotation:

$$\vec{A} = (\vec{U}\vec{V}^T)(\vec{V}\vec{D}\vec{V}^T)$$

$$= \underset{\text{rotation}}{\vec{R}} \quad \underset{\text{deformation}}{\vec{S}}$$