物理エンジンの概要

生い立ち、仕組み、特徴、上手な使い方

東京工業大学 未来産業技術研究所

長谷川晶一

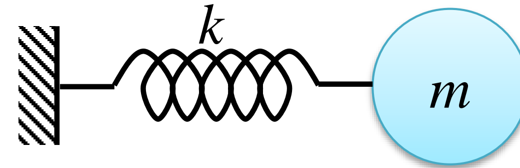
目次

- 解析解と物理シミュレーションと物理エンジンの違い
- 物理エンジンの生い立ち
- 物理エンジンの仕組み
  - 速度拘束をLCPに帰着させて解くタイプの物理エンジンの計算
  - 接触判定
- 安定性と制御
- 関節だけ高精度にシミュレーションする手法

解析解と物理シミュレーションと物理エンジンの違い

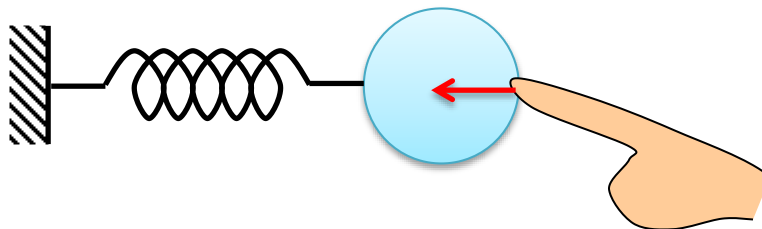
解析解と数値シミュレーション

- 解析
  - 運動方程式　$ma = f$
  - フックの法則　$f = -kx$



- $ma = -kx$ → 調和振動 → $x = \mathrm{A}\cos\omega t + \mathrm{B}\sin\omega t$ $\omega \equiv \sqrt{\frac{k}{m}}$

- 変化に対応できない



$f = -kx + f_c(t)$

$x = ???$

解析解と数値シミュレーション

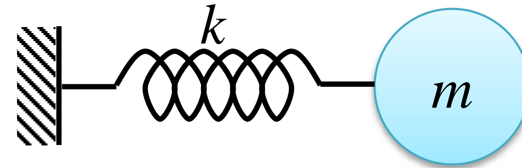
●数値シミュレーション

- $ma = -kx$

  $m\big(v(t+\Delta t) - v(t)\big) = -kx$

  $v(t+\Delta t) = -\frac{k}{m}x\,\Delta t$

- $x(t+\Delta t) = x(t) + v(t)\Delta t$



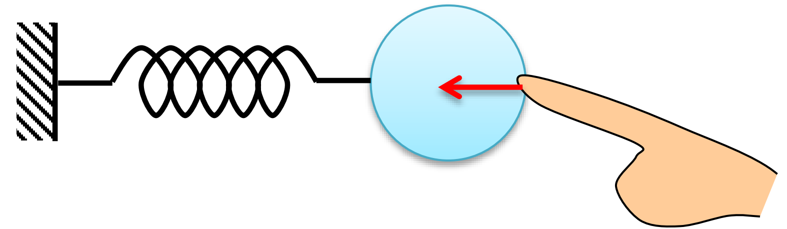
v(0)= 0	x(0)= 1	数値シミュレーション
v(0.1)=-1･0.1 +0=-0.1	x(0.1) = -0･0.1 + 1 = 1	
v(0.2)=-1･0.1 -0.1 = -0.2	x(0.2) = -0.1･0.1 + 1= 0.99	
v(0.3)= -0.99･0.1 -0.2 = -0.299	x(0.3) = -0.2･0.1 + 0.99= 0.97	

数値シミュレーション

- 変化に対応できる
  - $ma = -kx + f_c(t)$

$$m\big(v(t+\Delta t) - v(t)\big) = -kx + f_c(t)$$

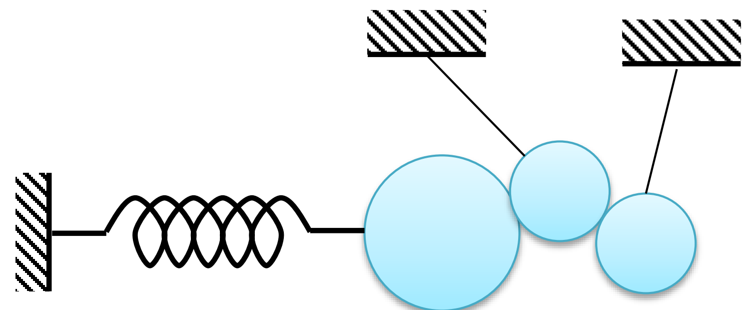
$$v(t+\Delta t) = -\frac{k}{m}x + f_c(t)\Delta t$$



v(0)= 0
v(0.1)=-1･0.1 +0=-0.1
v(0.2)=-1･0.1+f_c(0.2) -0.1 = -0.2+f_c(0.2)

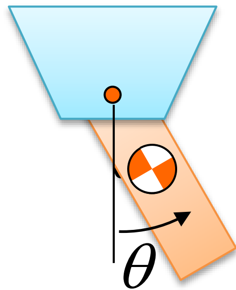
x(0)= 1
x(0.1) = -0･0.1 + 1 = 1
x(0.2) = -0.1･0.1 + 1= 0.99

- 変化する機構には対応できない
  - 機構毎に運動方程式が必要



物理シミュレーションと物理エンジンの違い

- 普通の物理シミュレーション
  - 運動方程式を人が立てる（解析力学を使うにしても）



$$(ml + I)\ddot{\theta} = -mg \sin\theta$$

↑ この式は人が見つける

$$L = K - U = \frac{1}{2}(ml + I)\dot{\theta}^2 - mg\cos\theta$$

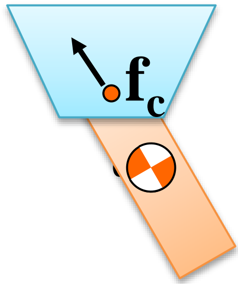
← 解析力学を使う場合でも
エネルギーの式は人が見つける

$$\frac{\partial}{\partial t}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = (ml + I)\ddot{\theta} + mg\sin\theta = 0$$

- 位置と速度の更新の式に変形し、シミュレーションする

物理シミュレーションと物理エンジンの違い

- 物理エンジン
  - 機構の設定から、運動方程式を自動的に立てる



$$m\dot{\boldsymbol{v}} = \boldsymbol{f_c} + m\boldsymbol{g}$$
$$\boldsymbol{I}\dot{\boldsymbol{\omega}} = \boldsymbol{r_c} \times \boldsymbol{f_c}$$

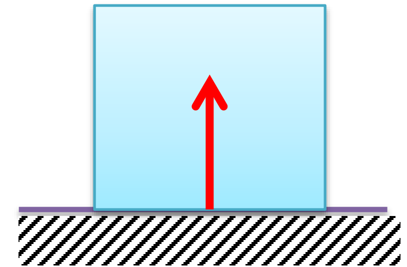
ニュートン･オイラーの運動方程式
＋ 拘束力$\boldsymbol{f_c}$

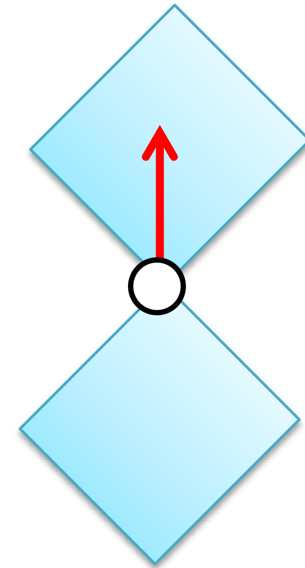
  - 拘束力$\boldsymbol{f_c}$を求めながら、剛体の位置、速度を更新する

- 接触など運動方程式の変化に対応できる

拘束力とは？

- 拘束を守らせる力
  - 物体の運動により決まる
- 例
  - 抗力
    - 物体が床に侵入しないように保つ力
  - 蝶番:軸回りの回転だけが可能
    - 位置と回転軸以外の向きを保つ力





- 他の外力は与えられている
  - 重量: $f_g = mg\ (g = 9.8m/s^2)$

物理エンジンの生い立ち

物理エンジンの生い立ち

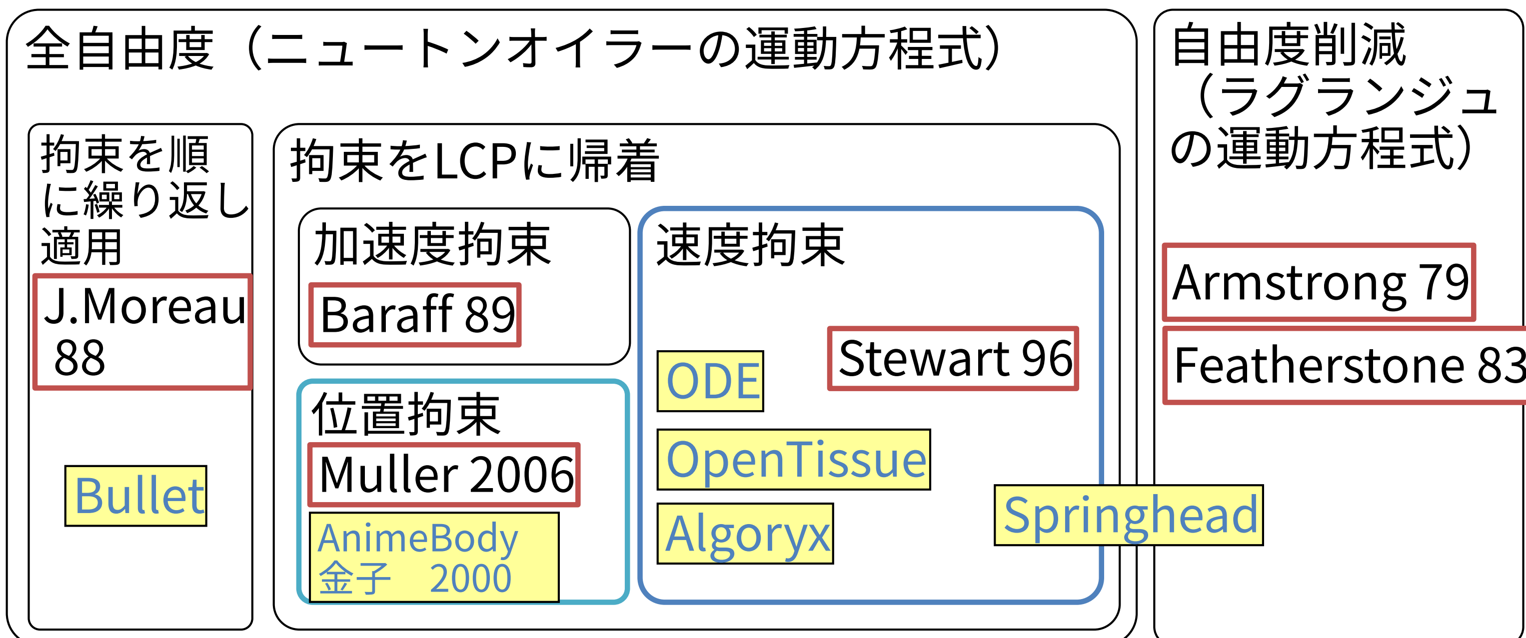
- 物理エンジン
  - 多数の剛体運動(マルチボディダイナミクス)のシミュレータ
- 剛体の運動
  - ニュートン力学(1736)、オイラーの運動方程式(1765)、ラグランジュの解析力学(1788)
- ADAMS (N. Orlandea et.al. 1977)
  - 剛体毎にラグランジュの運動方程式を立て、拘束力を求めながらシミュレーション
  - 最初のマルチボディダイナミクス解析ツール

マルチボディダイナミクスの色々

- 問題
  - 初期値問題、境界値問題、振動解析
- 運動方程式
  - 全自由度（ニュートン・オイラーの運動方程式）
  - 自由度削減（ラグランジュの運動方程式）
- 積分の仕方
  - 前進積分、後退積分、シンプレクティック、
- 拘束条件の書き方
  - 加速度の式
  - 速度の式
  - 位置の式
- 連立不等式の計算法
  - 直接法
  - LCP、QP
  - 拘束を順に解くことを繰り返す

マルチボディダイナミクスの色々

- 多体の剛体運動(マルチボディダイナミクス)のシミュレータ(初期値問題)の分類（論文とライブラリ）



物理エンジンの生い立ち

- アニメーション、ゲームへの応用

Ex:Armstrong 1979はロボット、1985はコンピュータグラフィクス

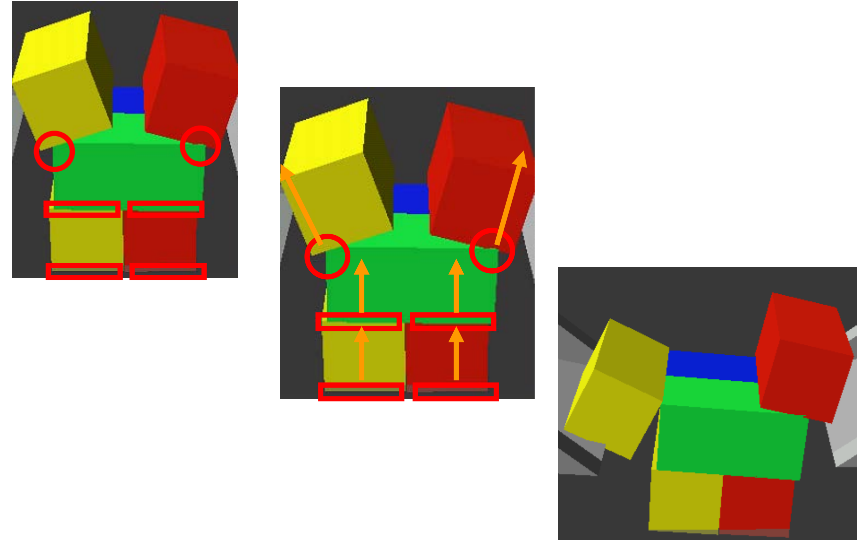
- ゲームの要求
  - リアルタイム性
  - 剛体数大、接触状態の変化が多い

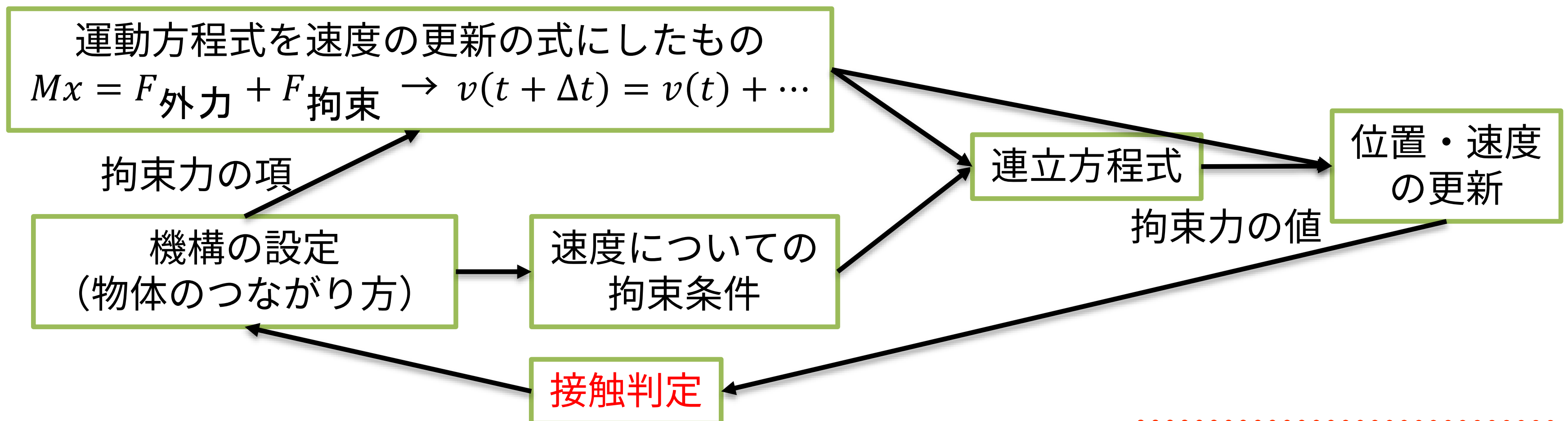
- 1997 Karma physics simulation engine (MathEngine)
  - 最初のゲーム向け物理エンジン（たぶん）
- 2000頃、ゲーム向け：Havok, Unreal2, PhysX, Natural Motion
  - 精度より速度
- 同じ頃、ロボット向け：Vortex, Algoryx
  - 精度も速度も

物理エンジンの仕組み

物理エンジンの計算の流れ

1. 接触を見つけ、拘束を更新
2. 拘束力を求める
3. 位置・速度を更新する

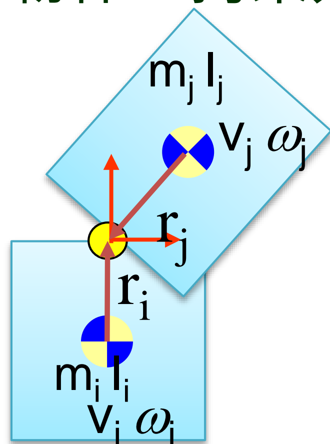




速度拘束をLCPに帰着させて解くタイプの物理エンジンの計算

剛体の運動方程式（ニュートン・オイラー）

- 2物体＋拘束力＋外力

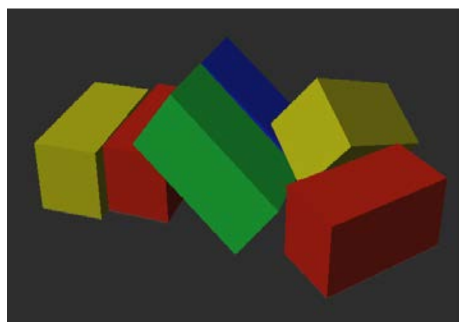


慣性 M　　速度 角速度 u　　拘束力 f_c　　その他の外力 重力等 f_e

$$\begin{bmatrix} m_i \boldsymbol{E}_{33} & 0 & 0 & 0 \\ 0 & I_i & 0 & 0 \\ 0 & 0 & m_j \boldsymbol{E}_{33} & 0 \\ 0 & 0 & 0 & \boldsymbol{I}_j \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{v}}_i \\ \dot{\boldsymbol{\omega}}_i \\ \dot{\boldsymbol{v}}_j \\ \dot{\boldsymbol{\omega}}_j \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_{ci} \\ \boldsymbol{\tau}_{ci} \\ \boldsymbol{f}_{cj} \\ \boldsymbol{\tau}_{cj} \end{bmatrix} + \begin{bmatrix} \boldsymbol{f}_{ei} \\ \boldsymbol{\tau}_{ei} \\ \boldsymbol{f}_{ej} \\ \boldsymbol{\tau}_{ej} \end{bmatrix}$$

$$\boldsymbol{E}_{33} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- たくさんの剛体＋拘束力＋外力



$$\boldsymbol{M}\dot{\boldsymbol{u}} = \boldsymbol{f}_c + \boldsymbol{f}_e$$

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M_1} & & & \\ & \boldsymbol{M_2} & & \\ & & \ddots & \\ & & & \boldsymbol{M_n} \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{\omega}_1 \\ \boldsymbol{v}_2 \\ \boldsymbol{\omega}_2 \\ \vdots \\ \boldsymbol{v}_n \\ \boldsymbol{\omega}_n \end{bmatrix} \quad \boldsymbol{f}_c = \begin{bmatrix} \boldsymbol{f}_{c1} \\ \boldsymbol{\tau}_{c1} \\ \boldsymbol{f}_{c2} \\ \boldsymbol{\tau}_{c2} \\ \vdots \\ \boldsymbol{f}_{cn} \\ \boldsymbol{\tau}_{cn} \end{bmatrix} \quad \boldsymbol{f}_e = \begin{bmatrix} \boldsymbol{f}_{e1} \\ \boldsymbol{\tau}_{e1} \\ \boldsymbol{f}_{e2} \\ \boldsymbol{\tau}_{e2} \\ \vdots \\ \boldsymbol{f}_{en} \\ \boldsymbol{\tau}_{en} \end{bmatrix}$$

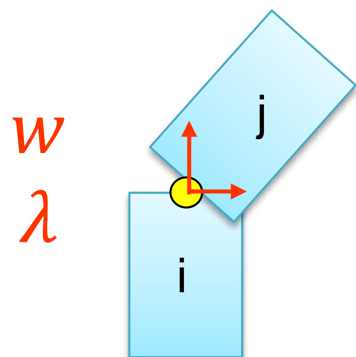
$$\boldsymbol{M}_i = \begin{bmatrix} m_i \boldsymbol{E}_{33} & \\ & \boldsymbol{I}_i \end{bmatrix}$$

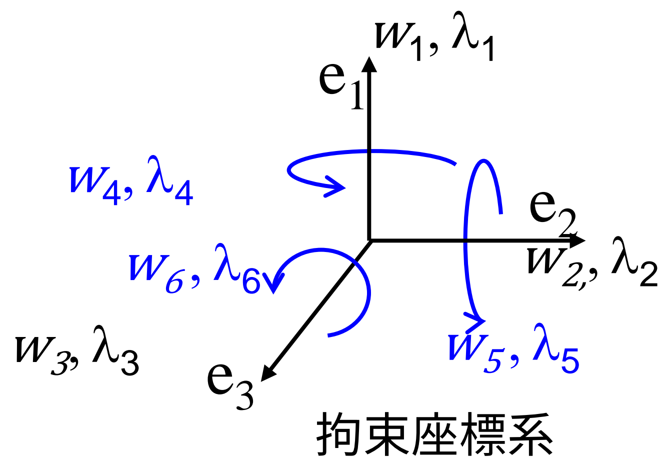
$$\begin{aligned} \boldsymbol{M}\dot{\boldsymbol{u}} &= \boldsymbol{f}_c + \boldsymbol{f}_e \\ \boldsymbol{u}[t+1] &= \boldsymbol{u}[t] + \boldsymbol{M}^{-1}\Delta t \boldsymbol{f}_c + \boldsymbol{M}^{-1}\Delta t \boldsymbol{f}_e \end{aligned}$$

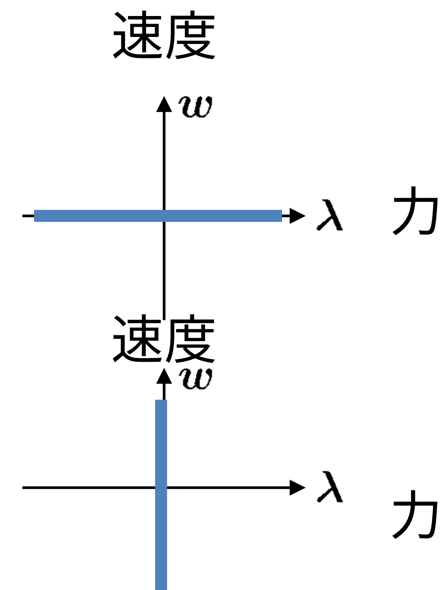
拘束条件（蝶番関節＝ヒンジジョイントの例）

w 関節位置での相対速度

λ 関節位置での拘束力と拘束トルク







- 蝶番の軸 e_3 だけが回転する

$$w_1, w_2, w_3, w_4, w_5 = 0$$

- 蝶番の軸 e_3は自由に回転＝トルクなし

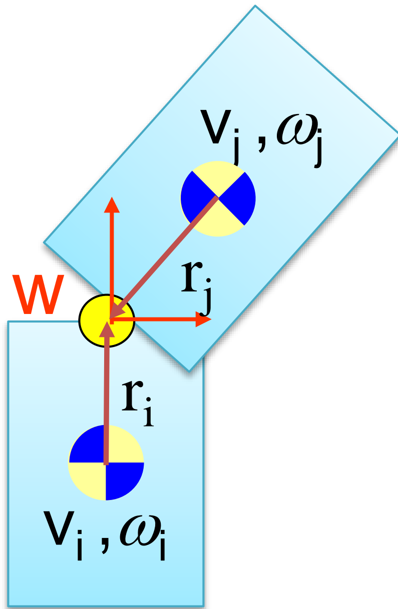
$$\lambda_6 = 0$$

拘束と運動方程式の連立方程式を解く

- 運動方程式を拘束座標系で表す → w,λ を使う

v_i, ω_i:物体 i の速度、角速度

$\boldsymbol{w}$:i からみた j の速度、角速度を拘束の座標系で書いたもの



$$
\begin{aligned}
w_1 &= \boldsymbol{e_1}\cdot((\boldsymbol{v_j}+\boldsymbol{\omega_j}\times\boldsymbol{r}_j)-(\boldsymbol{v_i}+\boldsymbol{\omega_i}\times\boldsymbol{r}_i)) \\
&= \boldsymbol{e_1^t}((\boldsymbol{v_j}-\boldsymbol{r}_j^\times\boldsymbol{\omega_j})-(\boldsymbol{v_i}-\boldsymbol{r}_i^\times\boldsymbol{\omega_i})) \\
&= \boldsymbol{e_1^t}\boldsymbol{v_j}-\boldsymbol{e_1^t}\boldsymbol{r}_j^\times\boldsymbol{\omega_j}-\boldsymbol{e_1^t}\boldsymbol{v_i}+\boldsymbol{e_1^t}\boldsymbol{r}_i^\times\boldsymbol{\omega_i} \\
&= \boldsymbol{e_1^t}\boldsymbol{v_j}+(\boldsymbol{r}_j^\times\boldsymbol{e}_1)^t\boldsymbol{\omega_j}-\boldsymbol{e_1^t}\boldsymbol{v_i}-(\boldsymbol{r}_i^\times\boldsymbol{e}_1)^t\boldsymbol{\omega_i} \\
&= \underbrace{\begin{bmatrix} -\boldsymbol{e}_1^t & -(\boldsymbol{r}_i^\times\boldsymbol{e}_1)^t & \boldsymbol{e}_1^t & (\boldsymbol{r}_j^\times\boldsymbol{e}_1)^t \end{bmatrix}}_{\mathrm{J_{row1}}} \underbrace{\begin{bmatrix} \boldsymbol{v}_i \\ \boldsymbol{\omega}_i \\ \boldsymbol{v}_j \\ \boldsymbol{\omega}_j \end{bmatrix}}_{\mathbf{u}}
\end{aligned}
$$

$$
\begin{aligned}
w_4 &= \boldsymbol{e_1}\cdot(\boldsymbol{\omega_j}-\boldsymbol{\omega_i}) \\
&= \boldsymbol{e_1^t}\boldsymbol{\omega_j}-\boldsymbol{e_1^t}\boldsymbol{\omega_i} \\
&= \underbrace{\begin{bmatrix} \mathbf{0}^t & -\boldsymbol{e_1^t} & \mathbf{0}^t & \boldsymbol{e_1^t} \end{bmatrix}}_{\mathrm{J_{row4}}} \underbrace{\begin{bmatrix} \boldsymbol{v}_i \\ \boldsymbol{\omega}_i \\ \boldsymbol{v}_j \\ \boldsymbol{\omega}_j \end{bmatrix}}_{\mathbf{u}}
\end{aligned}
$$

外積の行列表示

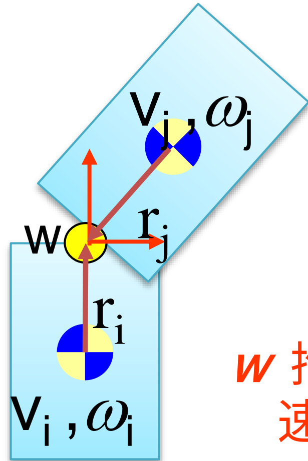
$$
\boldsymbol{a}\times\boldsymbol{b}=\boldsymbol{a}^\times\boldsymbol{b}
$$

$$
\boldsymbol{a}^\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}
$$

$w_2\ w_3,\ w_5\ w_6,$ も同様に書ける

拘束と運動方程式の連立方程式を解く

v_i, ω_i:物体 i の速度、角速度　　v_j, ω_j:物体 j の速度、角速度



$$
\boldsymbol{w} = \begin{bmatrix} -\boldsymbol{e}_1^t & -(\boldsymbol{r}_i^\times \boldsymbol{e}_1)^t & \boldsymbol{e}_1^t & (\boldsymbol{r}_j^\times \boldsymbol{e}_1)^t \\ -\boldsymbol{e}_2^t & -(\boldsymbol{r}_i^\times \boldsymbol{e}_2)^t & \boldsymbol{e}_2^t & (\boldsymbol{r}_j^\times \boldsymbol{e}_2)^t \\ -\boldsymbol{e}_3^t & -(\boldsymbol{r}_i^\times \boldsymbol{e}_3)^t & \boldsymbol{e}_3^t & (\boldsymbol{r}_j^\times \boldsymbol{e}_3)^t \\ \mathbf{0}^t & -\boldsymbol{e}_1^t & \mathbf{0}^t & \boldsymbol{e}_1^t \\ \mathbf{0}^t & -\boldsymbol{e}_2^t & \mathbf{0}^t & \boldsymbol{e}_2^t \\ \mathbf{0}^t & -\boldsymbol{e}_3^t & \mathbf{0}^t & \boldsymbol{e}_3^t \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_i \\ \boldsymbol{\omega}_i \\ \boldsymbol{v}_j \\ \boldsymbol{\omega}_j \end{bmatrix}
$$

$$
\boldsymbol{w} = \boldsymbol{J}\boldsymbol{u}
$$

w 拘束座標での速度・角速度

u 剛体座標の速度・角速度

$\boldsymbol{f}_j, \boldsymbol{\tau}_j$:i から j に加わる拘束力を剛体の座標系で書いたもの

$\boldsymbol{\lambda}$:i から j に加わる拘束力を拘束の座標系で書いたもの

$$
\begin{aligned}
\boldsymbol{f}_j &= \lambda_1 \boldsymbol{e}_1 + \lambda_2 \boldsymbol{e}_2 + \lambda_3 \boldsymbol{e}_3 \\
\boldsymbol{\tau}_j &= \boldsymbol{r}_j^\times (\lambda_1 \boldsymbol{e}_1 + \lambda_2 \boldsymbol{e}_2 + \lambda_3 \boldsymbol{e}_3) + \lambda_4 \boldsymbol{e}_1 + \lambda_5 \boldsymbol{e}_2 + \lambda_6 \boldsymbol{e}_3
\end{aligned}
$$

f_j, τ_j

λ

r_j

r_i

f_i, τ_i

$$
\begin{bmatrix} \boldsymbol{f}_i \\ \boldsymbol{\tau}_i \\ \boldsymbol{f}_j \\ \boldsymbol{\tau}_j \end{bmatrix} = \begin{bmatrix} -\boldsymbol{e}_1 & -\boldsymbol{e}_2 & -\boldsymbol{e}_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -\boldsymbol{r}_j^\times \boldsymbol{e}_1 & -\boldsymbol{r}_j^\times \boldsymbol{e}_2 & -\boldsymbol{r}_j^\times \boldsymbol{e}_3 & -\boldsymbol{e}_1 & -\boldsymbol{e}_2 & -\boldsymbol{e}_3 \\ \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{r}_j^\times \boldsymbol{e}_1 & \boldsymbol{r}_j^\times \boldsymbol{e}_2 & \boldsymbol{r}_j^\times \boldsymbol{e}_3 & \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 \end{bmatrix} \boldsymbol{\lambda}
$$

$$
\boldsymbol{f}_c = \boldsymbol{J}^t \boldsymbol{\lambda}
$$

f_c 剛体座標での力・トルク

l 拘束座標での力・トルク

拘束と運動方程式の連立方程式を解く

- 拘束を代入するために、*J*を使って運動方程式を変形する

$$
\begin{aligned}
M\dot{\boldsymbol{u}} &= \boldsymbol{f}_c + \boldsymbol{f}_e = \boldsymbol{J}^t\boldsymbol{\lambda} + \boldsymbol{f}_e \\
\boldsymbol{u}[t+1] &= \boldsymbol{u}[t] + M^{-1}\Delta t\boldsymbol{J}^t\boldsymbol{\lambda} + M^{-1}\Delta t\boldsymbol{f}_e \\
\boldsymbol{u}[t+1] &= \boldsymbol{u}[t] + \boldsymbol{M}^{-1}\Delta t\boldsymbol{J}^t\boldsymbol{\lambda} + \boldsymbol{M}^{-1}\Delta t\boldsymbol{f}_e \\
\boldsymbol{J}\boldsymbol{u}[t+1] &= \boldsymbol{J}\boldsymbol{u}[t] + \boldsymbol{J}\boldsymbol{M}^{-1}\Delta t\boldsymbol{J}^t\boldsymbol{\lambda} + \boldsymbol{J}\boldsymbol{M}^{-1}\Delta t\boldsymbol{f}_e \\
\boldsymbol{w}[t+1] &= \boldsymbol{w}[t] + \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^t\Delta t\boldsymbol{\lambda} + \boldsymbol{J}\boldsymbol{M}^{-1}\Delta t\boldsymbol{f}_e \\
\boldsymbol{w}[t+1] &= \boldsymbol{A}\boldsymbol{\lambda} + \boldsymbol{b}
\end{aligned}
$$

- 拘束条件と連立させる(蝶番の例)

拘束： $w_1, w_2, w_3, w_4, w_5 = 0$
$\lambda_6 = 0$

運動方程式に代入：

$$
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ w_6[t+1] \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ 0 \end{bmatrix} + \boldsymbol{b}
$$

← 連立方程式を解いて求める

速度・位置の更新

- 拘束力λを計算

$$\boldsymbol{w}[t+1] = \boldsymbol{A\lambda} + \boldsymbol{b} \quad \boldsymbol{A} = \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^t\Delta t$$
$$\boldsymbol{b} = \boldsymbol{w}[t] + \boldsymbol{J}\boldsymbol{M}^{-1}\Delta t \boldsymbol{f}_e$$

- 剛体の速度を更新

$$\boldsymbol{u}[t+1] = \boldsymbol{u}[t] + \boldsymbol{M}^{-1}\Delta t \boldsymbol{J}^t \boldsymbol{\lambda} + \boldsymbol{M}^{-1}\Delta t \boldsymbol{f}_e$$

- 剛体の位置を更新

$$\boldsymbol{s}[t+1] = \boldsymbol{s}[t] + \boldsymbol{S}\Delta t \boldsymbol{u}[t+1]$$

全剛体の位置姿勢

$$\boldsymbol{s} = \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{q}_1 \\ \boldsymbol{r}_2 \\ \boldsymbol{q}_2 \\ \vdots \\ \boldsymbol{r}_n \\ \boldsymbol{q}_n \end{bmatrix}$$

位置

$$\boldsymbol{r_i} = \begin{bmatrix} r_{ix} \\ r_{iy} \\ r_{iz} \end{bmatrix}$$

姿勢 Quaternion

$$\boldsymbol{q_i} = \begin{bmatrix} r_{iw} \\ r_{ix} \\ r_{iy} \\ r_{iz} \end{bmatrix}$$

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{E}_{33} & & & & \\ & \boldsymbol{Q}_1 & & & \\ & & \ddots & & \\ & & & \boldsymbol{E}_{33} & \\ & & & & \boldsymbol{Q}_n \end{bmatrix}$$

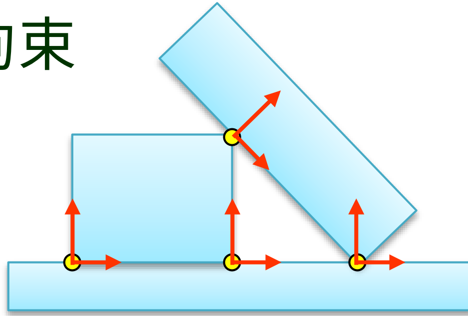
速度をQuaternionの微分に変換する行列

$$\dot{\boldsymbol{q}}_i = \boldsymbol{Q}_i \boldsymbol{\omega}_i$$

$$\boldsymbol{Q_i} = \frac{1}{2}\begin{bmatrix} -x_i & -y_i & -z_i \\ w_i & z_i & -y_i \\ -z_i & w_i & x_i \\ y_i & -x_i & w_i \end{bmatrix}$$

接触の拘束

●接触拘束



$\boldsymbol{w}$　接触点の座標系での相対速度

$\boldsymbol{\lambda}$　接触点の座標系での接触力

- 法線方向：お互いに侵入しない＝
  侵入時は接触力が働き速度0、離れるときは接触力が0

$$(\lambda_1 > 0) \wedge (w_1 = 0) \quad or \quad (\lambda_1 = 0) \wedge (w_1 > 0)$$

- 接線方向：静止摩擦／動摩擦：

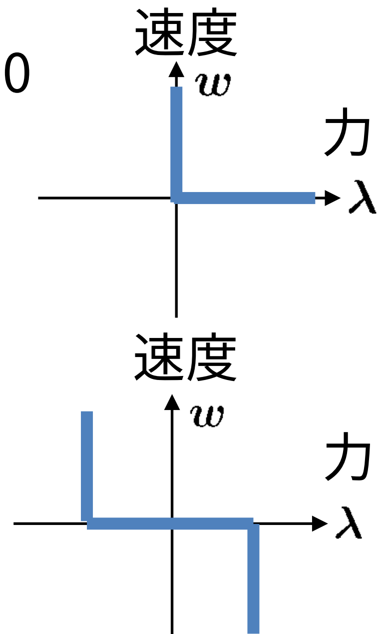
$$(-\mu\lambda_1 < \lambda_2, \lambda_3 < \mu\lambda_1) \wedge (w_2, w_3 = 0) \quad or$$

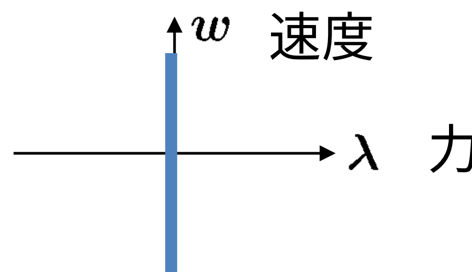
$$(\lambda_2 = \mu\lambda_1 \wedge w_2 > 0) \vee (\lambda_2 = -\mu\lambda_1 \wedge w_2 < 0),$$

$$(\lambda_3 = \mu\lambda_1 \wedge w_3 > 0) \vee (\lambda_3 = -\mu\lambda_1 \wedge w_3 < 0)$$

- トルクは0：

$$\lambda_4, \lambda_5, \lambda_6 = 0$$



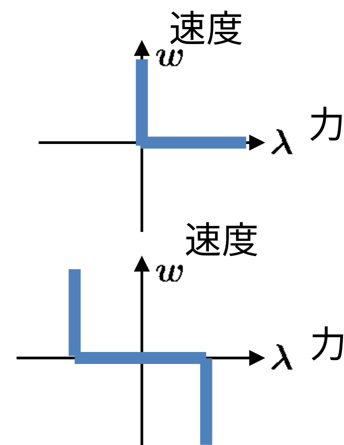


拘束条件を連立

- 拘束座標系に変換した運動方程式に、

$$\boldsymbol{w}[t+1] \quad = \quad \boldsymbol{A\lambda} + \boldsymbol{b}$$

- 拘束条件を代入



$$(\lambda_1 > 0) \wedge (w_1 = 0) \quad or \quad (\lambda_1 = 0) \wedge (w_1 > 0)$$
$$(-\mu\lambda_1 < \lambda_2, \lambda_3 < \mu\lambda_1) \wedge (w_2, w_3 = 0) \quad or$$
$$(\lambda_2 = \mu\lambda_1 \wedge w_2 > 0) \vee (\lambda_2 = -\mu\lambda_1 \wedge w_2 < 0),$$
$$(\lambda_3 = \mu\lambda_1 \wedge w_3 > 0) \vee (\lambda_3 = -\mu\lambda_1 \wedge w_3 < 0)$$

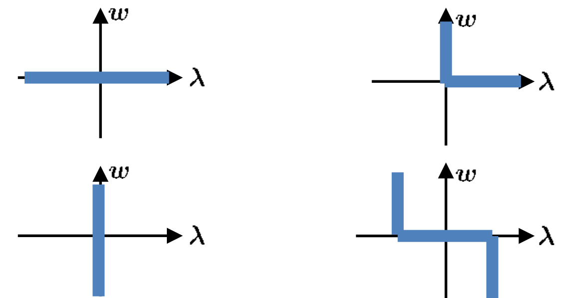
$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{b}$$

を解いてみて、
λ_i が $w_i = 0$ の条件から外れた場合、
$\lambda_i =$ 固定値 として w_i を求める。

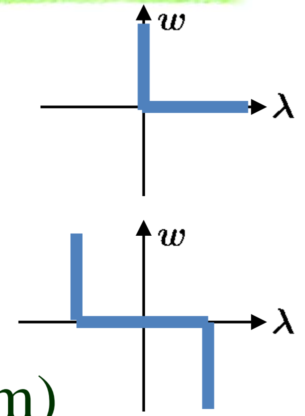
まとめ

- 物理エンジンでは、拘束を速度ベースで定式化し、運動方程式(速度の更新式)と連立させて、LCPにして解く方法が良く使われる。
- 具体的な計算
  - 速度の更新式をヤコビアンJで座標変換 → wとλの運動方程式
  - 拘束を相対速度wと拘束力λについての式に定式化 wとλの運動方程式に代入 → LCPになる
  - LCPを解いてλを求め、剛体の速度・位置を更新

- 拘束
  - 関節は直線、接触は折れ線
  - ２変数あるように見えて実質１変数



連立方程式の解き方



- 拘束を組み込んだ運動方程式

$$\boldsymbol{w}[t+1] \quad = \quad \boldsymbol{A\lambda} + \boldsymbol{b}$$

式1本で変数が2つ

＋ w_i と λ_i の相補条件（片方が動くとき片方は固定)

線形相補問題 (LCP:Linear Complementary Problem)

- LCPを解く
  - ピボッティング法　・・・　厳密．低速
  - 反復解法　・・・　高速．精度は反復回数に依る
    - 行列分割法
      - ヤコビ法
      - **ガウス・ザイデル法**
    - ニュートン法

ガウスザイデル法・SOR法

●行列分割法

- 巨大な連立方程式の近似解を少ない計算で解く方法のひとつ

$$
\begin{aligned}
\boldsymbol{w}[t+1] &= \boldsymbol{A}\boldsymbol{\lambda}+\boldsymbol{b} \\
\boldsymbol{w}[t+1] &= (\boldsymbol{D}-\boldsymbol{F})\boldsymbol{\lambda}+\boldsymbol{b} \\
\boldsymbol{D}\boldsymbol{\lambda} &= \boldsymbol{F}\boldsymbol{\lambda}+\boldsymbol{w}[t+1]-\boldsymbol{b} \\
\boldsymbol{\lambda} &= \boldsymbol{D}^{-1}\boldsymbol{F}\boldsymbol{\lambda}+\boldsymbol{D}^{-1}(\boldsymbol{w}[t+1]-\boldsymbol{b})
\end{aligned}
$$

もし、漸化式

$$\boldsymbol{\lambda}^{i+1} = \boldsymbol{D}^{-1}\boldsymbol{F}\boldsymbol{\lambda}^{i}+\boldsymbol{D}^{-1}(\boldsymbol{w}[t+1]-\boldsymbol{b})$$

が収束するなら

$$\boldsymbol{\lambda}^{\infty} = \boldsymbol{D}^{-1}\boldsymbol{F}\boldsymbol{\lambda}^{\infty}+\boldsymbol{D}^{-1}(\boldsymbol{w}[t+1]-\boldsymbol{b})$$

なので、求めたい $\boldsymbol{\lambda}=\boldsymbol{\lambda}^{\infty}$ となる。

Aが正定値対称行列なら収束する
細長いもの・慣性テンソルが小さいものは収束しにくい

- Dは対角行列のような簡単な行列にする。
- DをAの対角成分とし、残りをFとしたのがヤコビ法
- それを工夫したのがガウスザイデル法

ガウスザイデル法・SOR法

- ガウスザイデル法

$$\boldsymbol{\lambda}_{i+1} = \boldsymbol{D}^{-1}\boldsymbol{F}\boldsymbol{\lambda}_i + \boldsymbol{w}[t+1] - \boldsymbol{b}$$

ヤコビ法：λを一度に更新

$$\begin{bmatrix} \lambda_1^{i+1} \\ \lambda_2^{i+1} \\ \lambda_3^{i+1} \\ \vdots \\ \lambda_n^{i+1} \end{bmatrix} = \boldsymbol{D}^{-1}\boldsymbol{F} \begin{bmatrix} \lambda_1^{i} \\ \lambda_2^{i} \\ \lambda_3^{i} \\ \vdots \\ \lambda_n^{i} \end{bmatrix} + \boldsymbol{D}^{-1}\boldsymbol{w}[t+1] - \boldsymbol{D}^{-1}\boldsymbol{b}$$

ガウス・ザイデル法：λを1行ずつ更新

$$\begin{bmatrix} \lambda_1^{i+1} \\ \lambda_2^{i+1} \\ \lambda_3^{i+1} \\ \vdots \\ \lambda_n^{i+1} \end{bmatrix} = \boldsymbol{D}^{-1}\boldsymbol{F} \begin{bmatrix} \lambda_1^{i+1} \\ \lambda_2^{i+1} \\ \lambda_3^{i} \\ \vdots \\ \lambda_n^{i} \end{bmatrix} + \boldsymbol{D}^{-1}\boldsymbol{w}[t+1] - \boldsymbol{D}^{-1}\boldsymbol{b}$$

更新済み

更新中

未更新

- SOR(Successive Over-Relaxation)法
  - ガウスザイデル法をさらに加速

ガウスザイデル法

$$\lambda_j^{i+1} = \boldsymbol{D}^{-1}\boldsymbol{F}_j \begin{bmatrix} \lambda_1^{i+1} \\ \vdots \\ \lambda_n^{i} \end{bmatrix} + \boldsymbol{D}_j^{-1} w_j[t+1] - \boldsymbol{D}_j^{-1} b_j$$

1.6倍加速

$$\lambda_j^{i+1} = 1.6\left(\left(\boldsymbol{D}^{-1}\boldsymbol{F}_j \begin{bmatrix} \lambda_1^{i+1} \\ \vdots \\ \lambda_n^{i} \end{bmatrix} + \boldsymbol{D}_j^{-1} w_j[t+1] - \boldsymbol{D}_j^{-1} b_j\right) - \lambda_j^i\right) + \lambda_j^i$$

  - 早く収束することもある

LCPとガウスザイデル法

●ガウスザイデル法は1行ずつ更新

$$\begin{bmatrix}\lambda_1^{i+1}\\ \lambda_2^{i+1}\\ \lambda_3^{i+1}\\ \vdots\\ \lambda_n^{i+1}\end{bmatrix} = \boldsymbol{D}^{-1}\boldsymbol{F}\begin{bmatrix}\lambda_1^{i+1}\\ \lambda_2^{i+1}\\ \lambda_3^{i}\\ \vdots\\ \lambda_n^{i}\end{bmatrix} + \boldsymbol{w}[t+1] - \boldsymbol{b}$$

更新済み

更新中

未更新

●接触力をうまく計算できる

- お互いに侵入しない＝力が働いて止まる 又は 力は働かず離れる:

$$(\lambda_1 > 0) \wedge (w_1 = 0) \quad or \quad (\lambda_1 = 0) \wedge (w_1 > 0)$$

λ_1を更新するたびに、$\lambda_1>0$をチェック 負になったら $\lambda_1=0$に設定

- 静止摩擦か動摩擦:

$$(-\mu\lambda_1 < \lambda_2, \lambda_3 < \mu\lambda_1) \wedge (w_2, w_3 = 0) \quad or$$

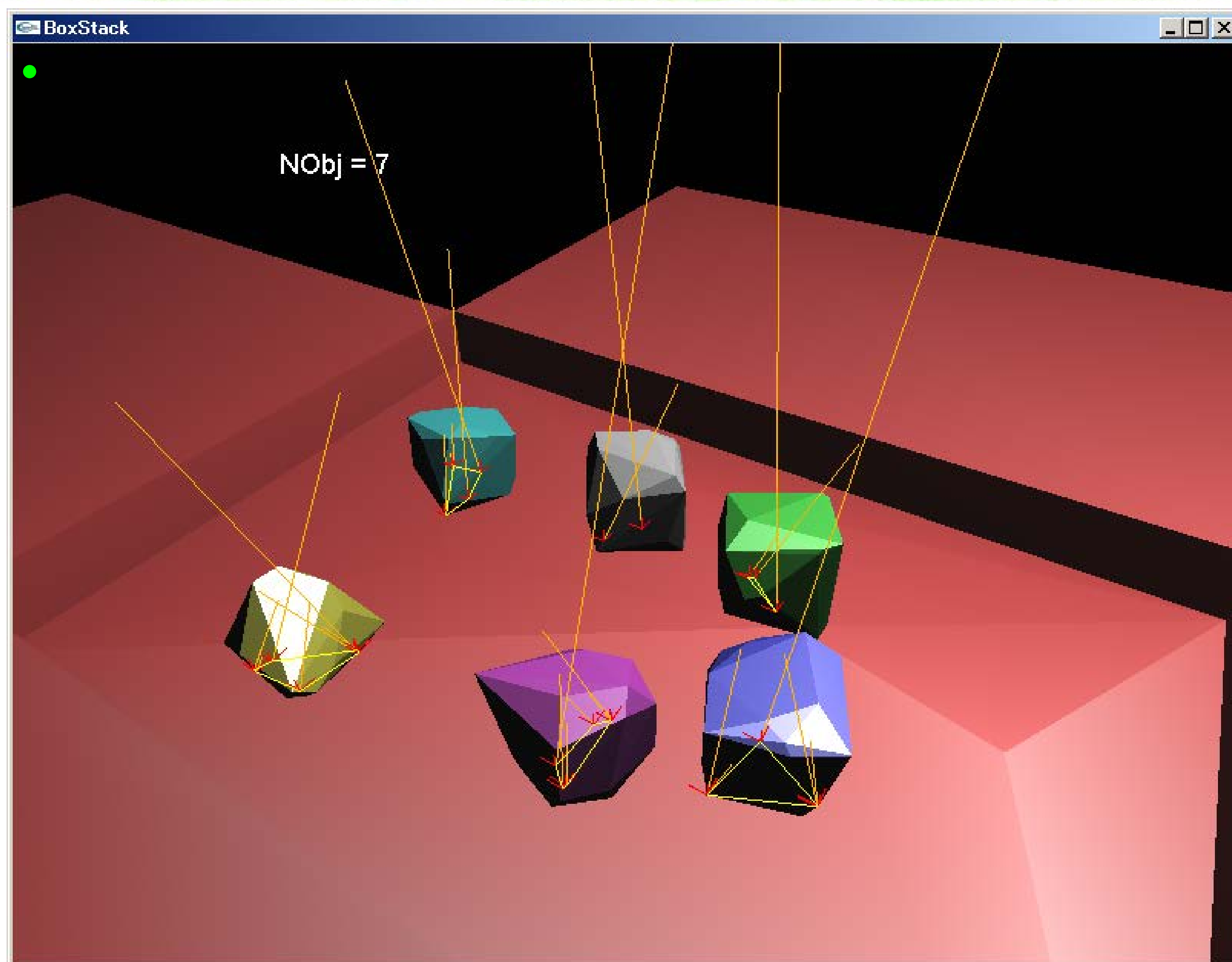
$$(\lambda_2 = \mu\lambda_1 \wedge w_2 > 0) \vee (\lambda_2 = -\mu\lambda_1 \wedge w_2 < 0),$$

$$(\lambda_3 = \mu\lambda_1 \wedge w_3 > 0) \vee (\lambda_3 = -\mu\lambda_1 \wedge w_3 < 0)$$

更新したλ_1を使って $-\mu\lambda_1 < \lambda_2 < \mu\lambda_1$をチェック

はみ出したら、$\lambda_1 = +-\mu\ \lambda_1$でクリップ

シミュレーションの様子



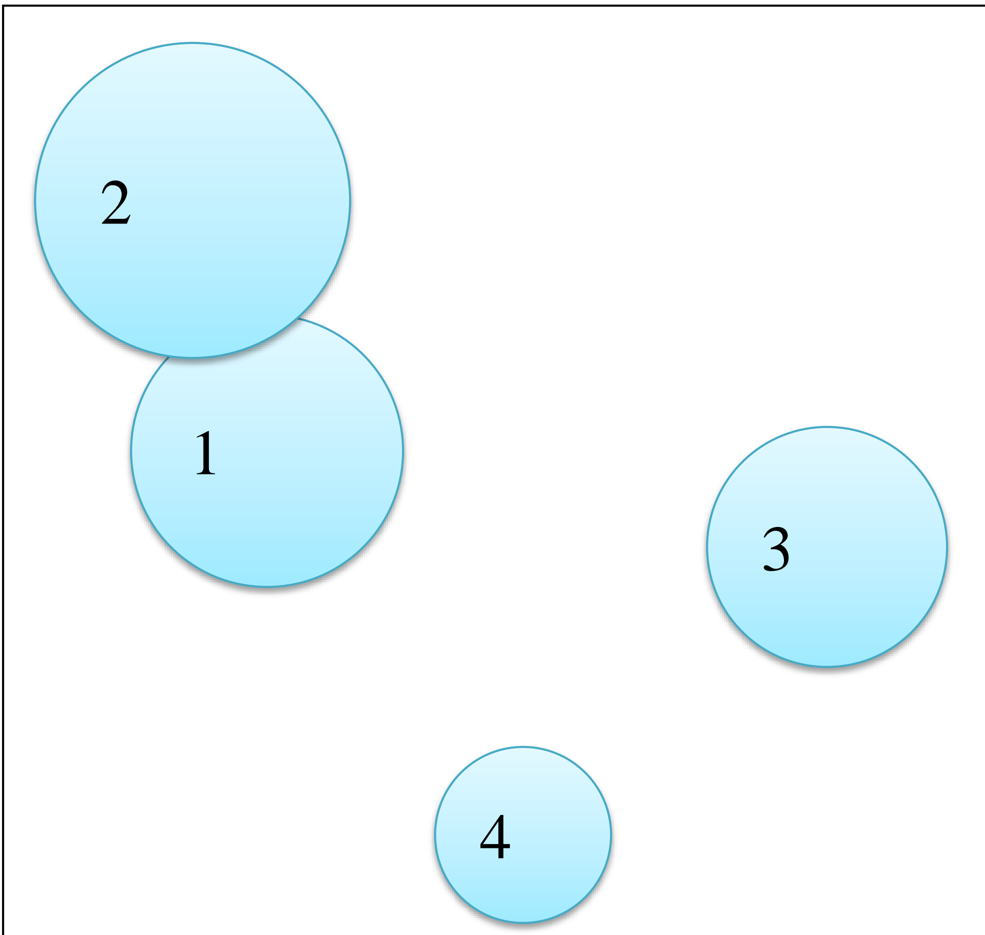
接触判定

接触判定

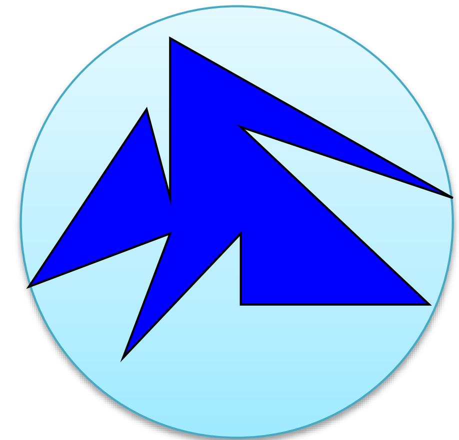
- Broad phase(大まかな枝刈り処理)
  - 接触判定とソート
    - 総当り と Sweep and prune
- Narrow phase（ポリゴン同士の判定など）のアルゴリズム
  - 物理シミュレーションに必要な接触情報
  - 接触点と法線
    - 最小と極小
    - GJK，Lin-Canny と 探索

接触判定とソート

- Broad phase
  - Bounding Volume 同士の大まかな枝刈り

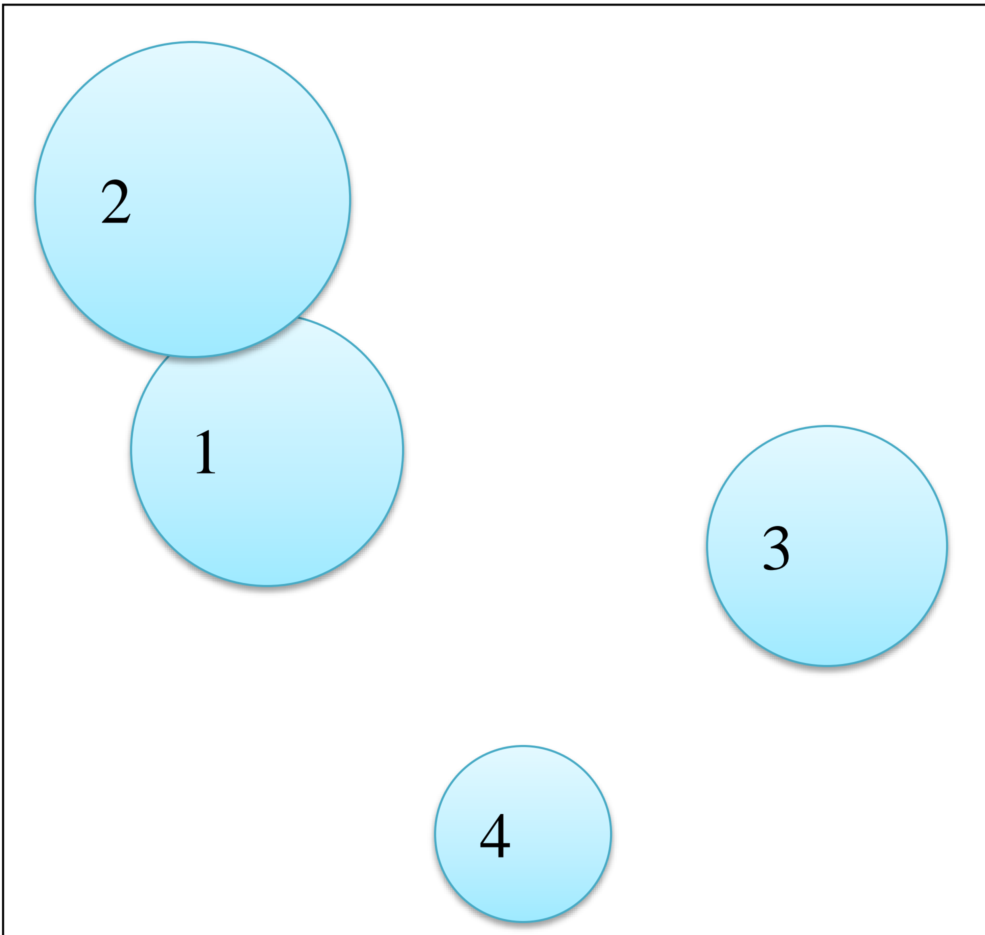


Bounding volume



総当り法

- すべてのペアをチェック
  - n・(n-1)/2 個のペアをチェック



1-2
1-3
1-4
2-3
2-4
3-4

$$\frac{n \cdot (n-1)}{2}$$回

重なっているかチェック

Sweep and prune

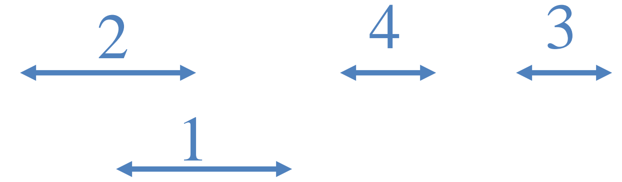
- 最初にある軸についてソート
  - 重なっている可能性のあるものだけを判定

ソート

2b 1b 2e 1e 4b 4e 3b 3e

左から重なりをチェック

2b 1b 2e 1e 4b 4e 3b 3e



1-2

n回チェック

ソート+n回チェック

ソートのアルゴリズム

- Selection sort　（総当り）

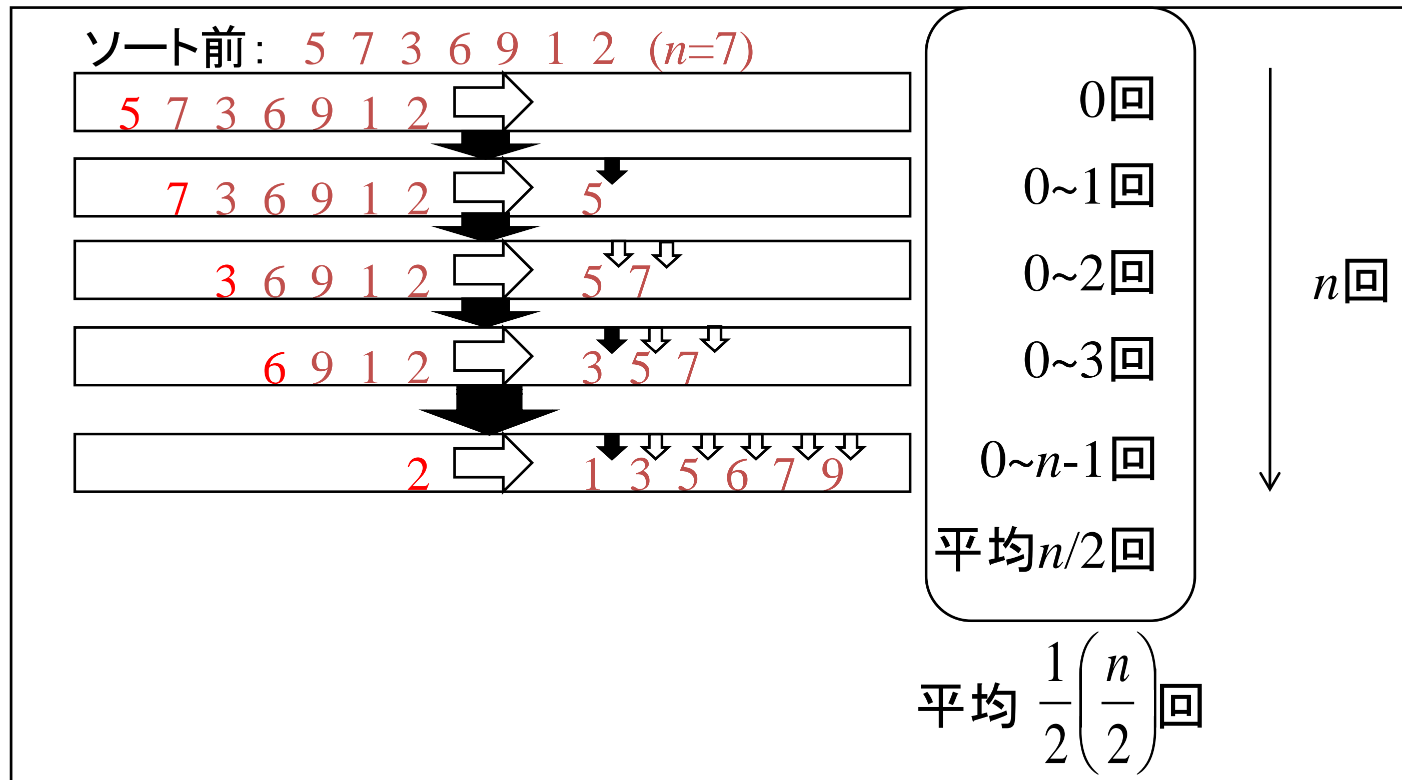
  $O(n^2)$

- Quick sort

  $O(n \log n)$

Sweep and prune に最適なソート

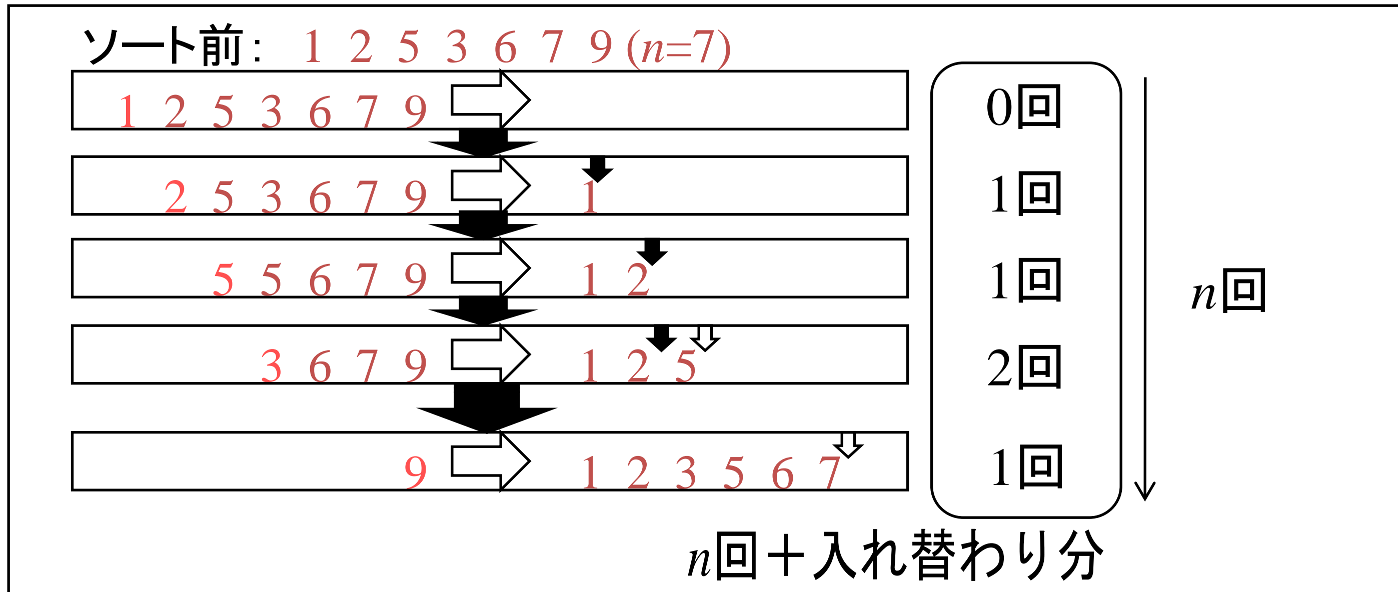
- 物理シミュレーション → 物体は，それほど入れ替わらない
- 　→ ほとんどソート済み → Insert sort が早い
- Insert sort 通常時：



$$\frac{1}{2}\left(\frac{n}{2}\right)\times n = \frac{n^2}{4} \qquad O(n^2)$$

Sweep and prune に最適なソート

- Insert sort ほとんどソート済みの場合：



$$O(n+\text{入れ替わり分}) = O(n+\alpha n) = O(n)$$

（α :隣との入れ替わりが起こる割合）

- 参考：ランダムに入れ替わる場合：

$$O(n+\text{入れ替わり分}) = O(n+\alpha n\cdot\beta n) = O(n^2)$$

α：入れ替わりが起こる割合,

β：入れ替わり先の位置の比

1 2 7 5 6 3 9

$$\beta = \frac{3}{7}$$

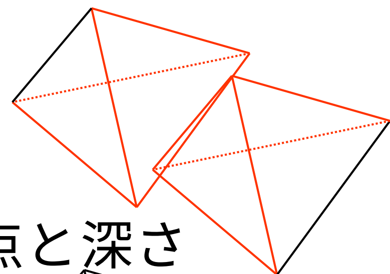
Sweep and pruneの計算量

- 総当りの判定　→　$O(n^2)$
- Sweep and prune →　$O($ソート＋n回チェック$)$
  - Insertion sortを使うと→　$O(n)$

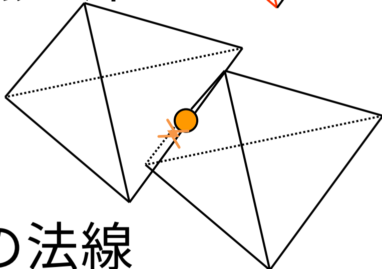
- Sweep and pruneとソート
  - 物体間に < 演算 を導入
  - 高速なソートのアルゴリズムを利用
  - 前回の計算結果を利用

物理エンジンに必要な接触情報

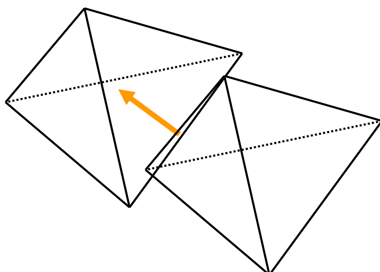
- Narrow phase：具体的な形状（プリミティブ・ポリゴン）同士の判定
  - 接触情報の種類
    - 接触したポリゴン



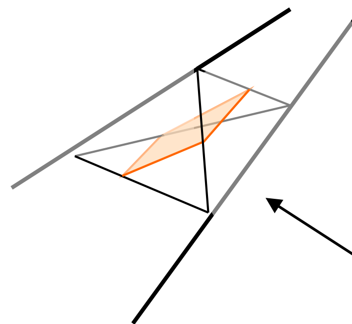
    - 接触点と深さ



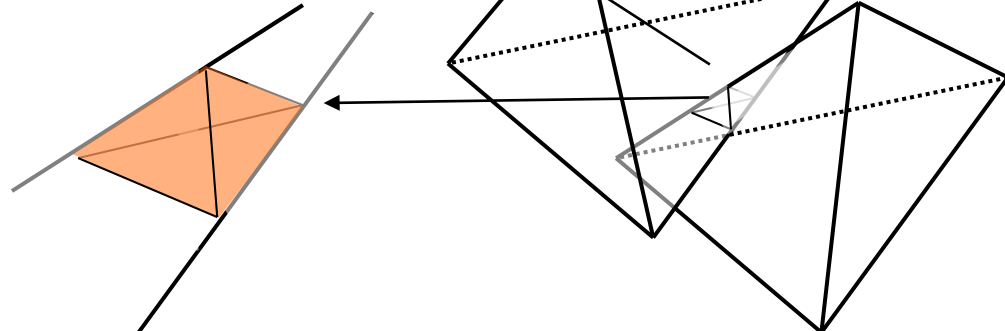
    - 接触の法線



    - 接触領域（面）の形状

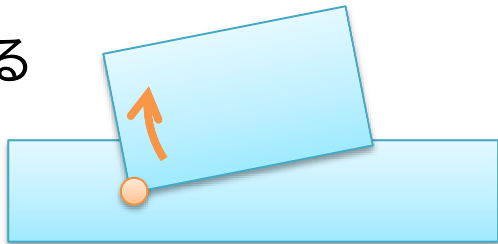


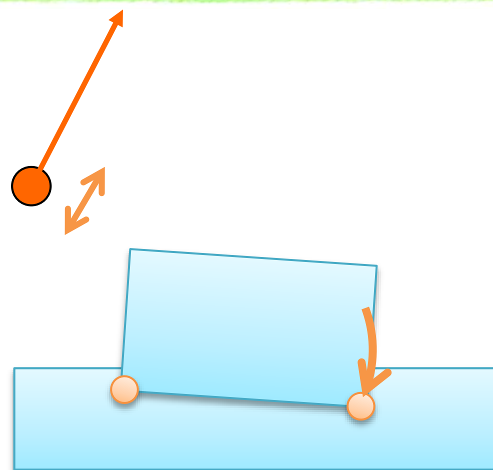
    - 接触部分（立体）の形状



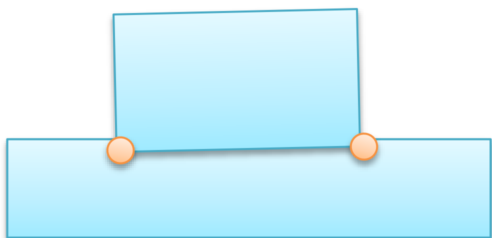
接触情報と物理エンジンの挙動

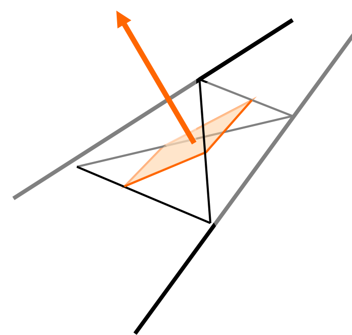
- 接触点と深さ＋法線
  - 一番深い点に接触拘束をつける
  - 揺れる



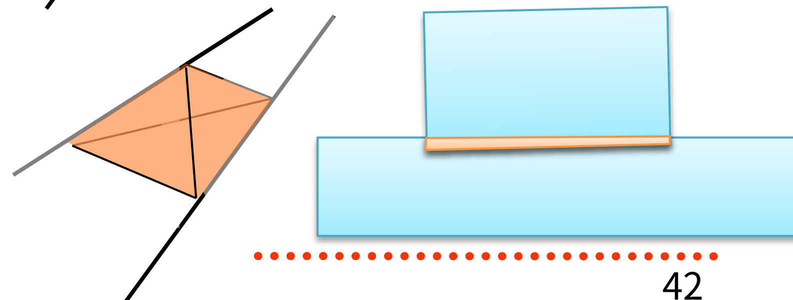


- 接触領域の頂点と深さ＋法線
  - 安定



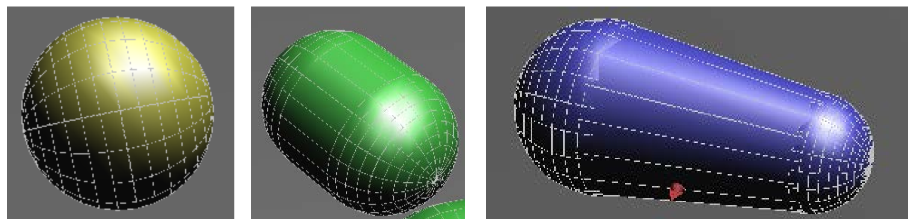


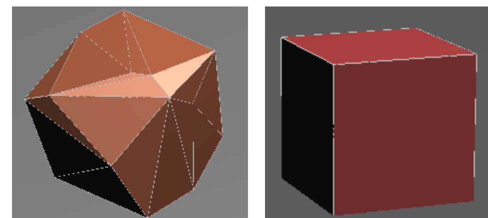
- 接触領域の体積（形状）＋法線
  - 摩擦力の分布なども計算可能

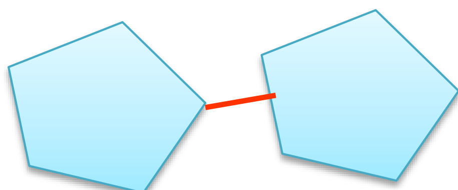


接触点と法線を求める

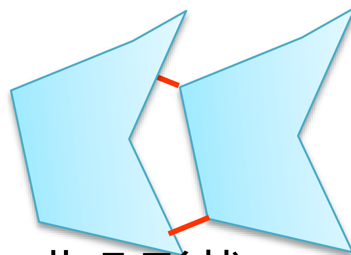
- 形状の表現
  - プリミティブ、 凸ポリゴン



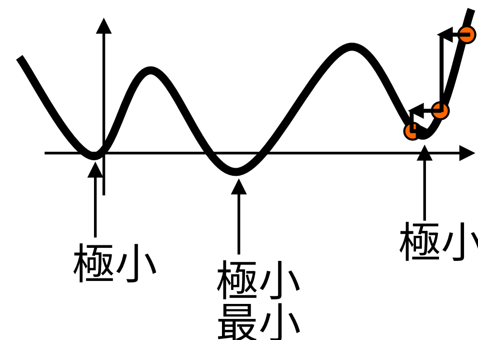




凸形状



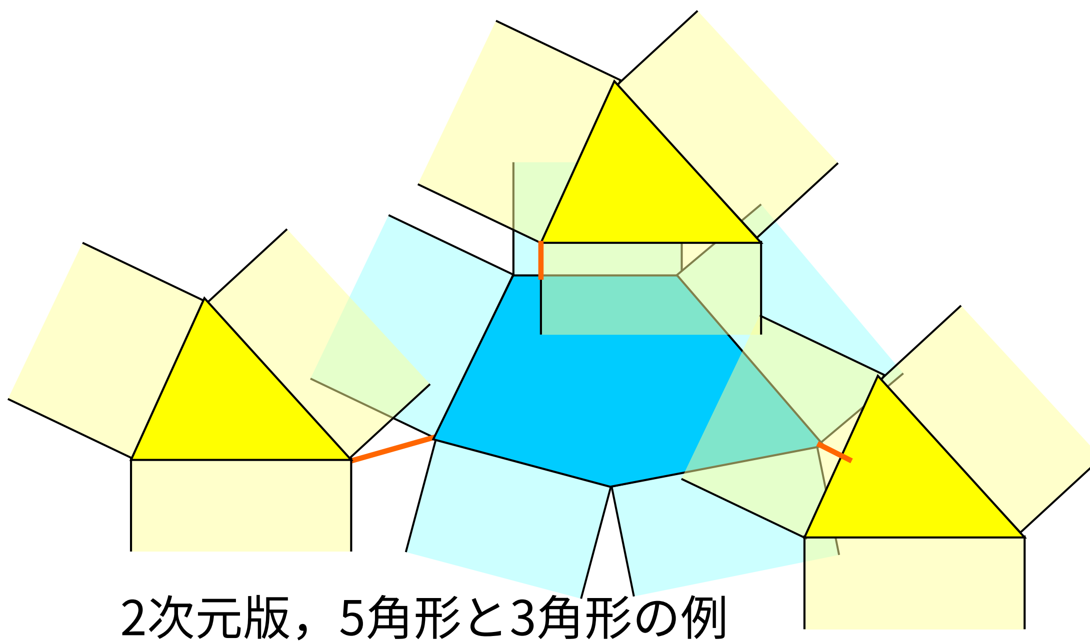
非凸形状



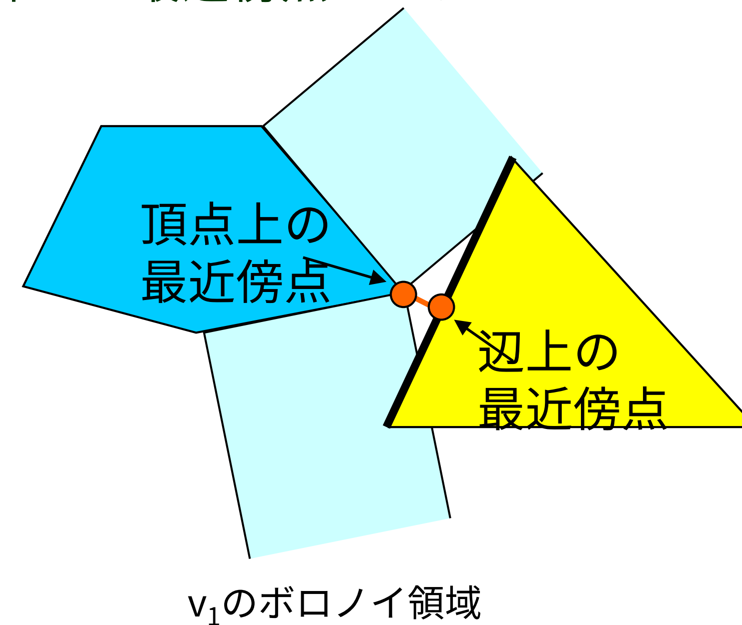
- 凸形状 距離が極小となる点が1点 → 最近傍点が簡単に求まる
  - Lin-Canny algorithm (1991)
  - GJK(Gilbert–Johnson–Keerthi) algorithm (1988)
- 非凸ポリゴン同士を扱える物理エンジンは少ない。
  - 予め凸分割した方が速いので

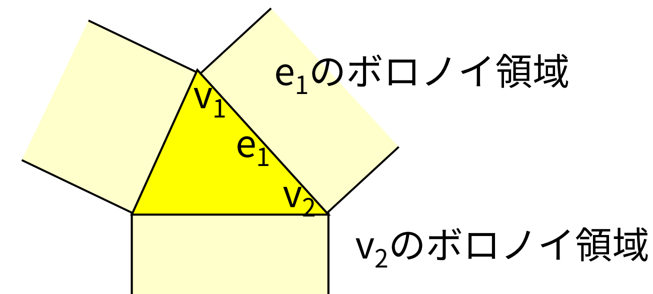
Lin-Canny algorithm

- 頂点/辺/面のボロノイ領域に，もう一つの凸多面体上の頂点/辺/面上の最近傍点が含まれていれば，そのペアが2凸多面体上の最近傍点のペア



2次元版，5角形と3角形の例

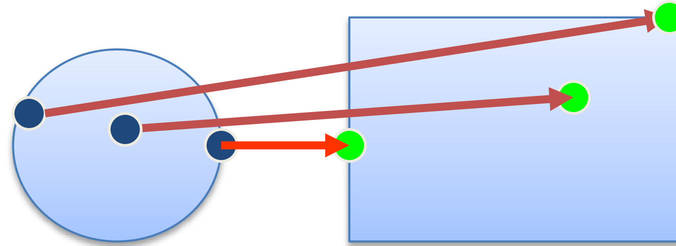




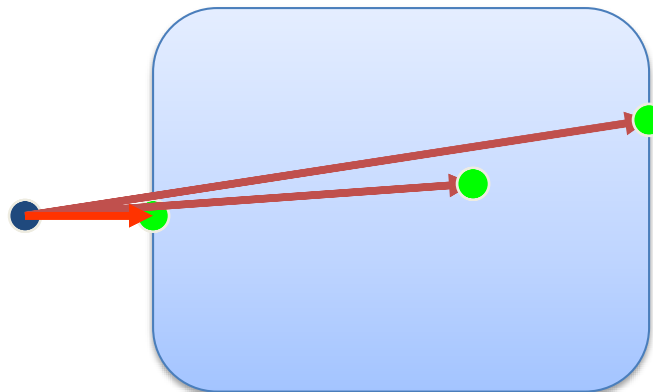
- 含まれていなければ，
より近くなる隣の頂点/辺/面へ移動．

GJK

- 凸形状A上の点から，凸形状B上の点へのベクトルを原点を始点に並べる（Minkowski sum）とベクトルの終点の集合も凸形状になる



元の図形

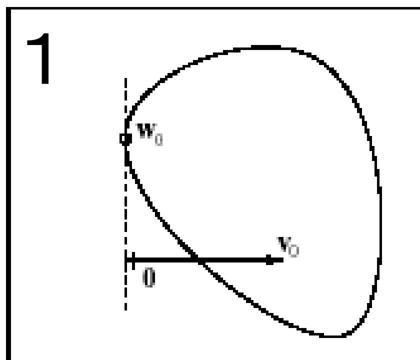


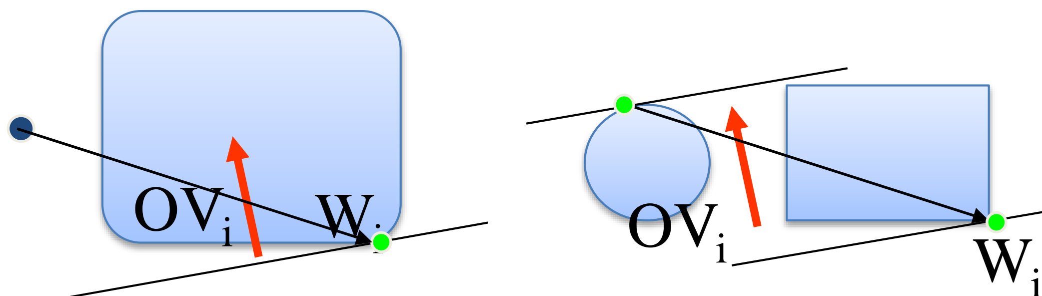
Minkowski Sum をとったもの

GJK

V_0 : 凸形状内の任意の点

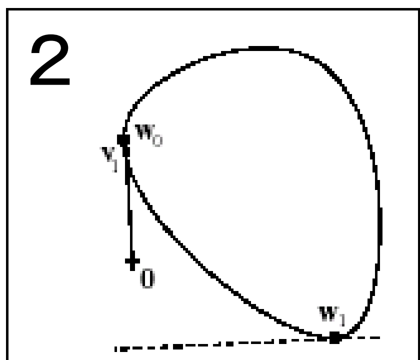
W_i :$\overrightarrow{OV_i}$と$\overrightarrow{OW_i}$の内積が最小の点(support point)

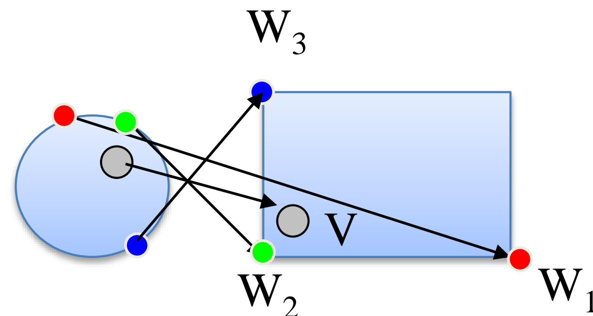


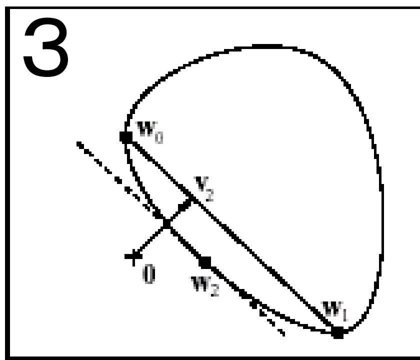


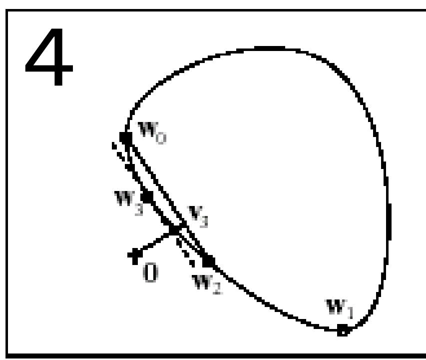
V_i :三角形W_{i-2} W_{i-1} W_i内の点で原点に一番近い点

$V_i = s\,W_{i-2} + t\,W_{i-1} + (1-s-t)\,W_i$





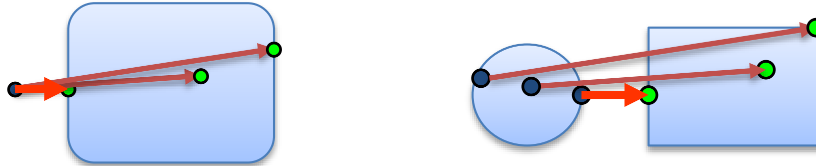




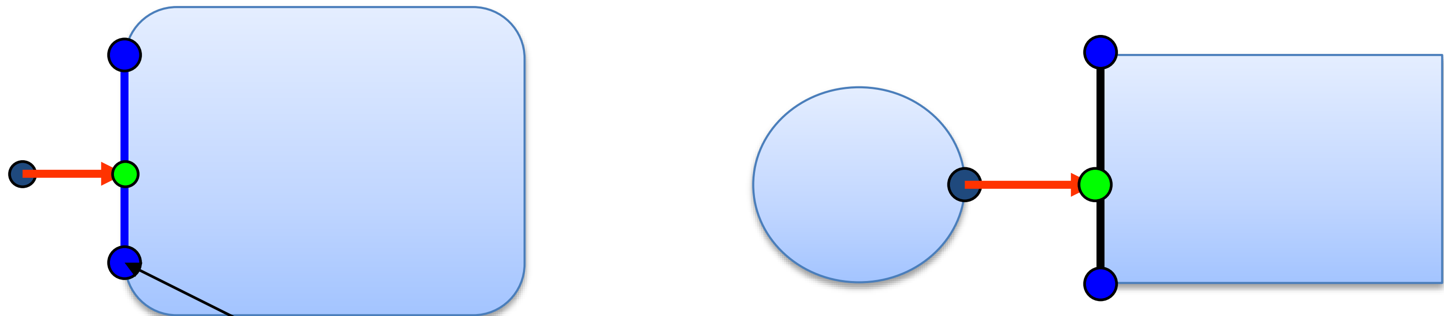
元の図形上で、最近傍点が求まる

GJKの結果の解釈

- 法線 そのままもとの世界の法線



- 接触点　Support point を内分した点
  →実世界の Support points を内分して求める．



これを求める→実世界の頂点の引き算
どれを使ったか覚えておけば元に戻せる

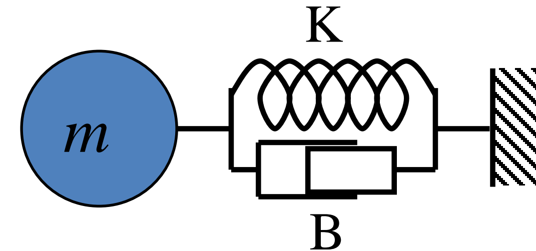
安定性と制御

安定性と積分の方法

●前進（Explicit）積分と後退（Implicit）積分

●バネ・ダンパの運動方程式

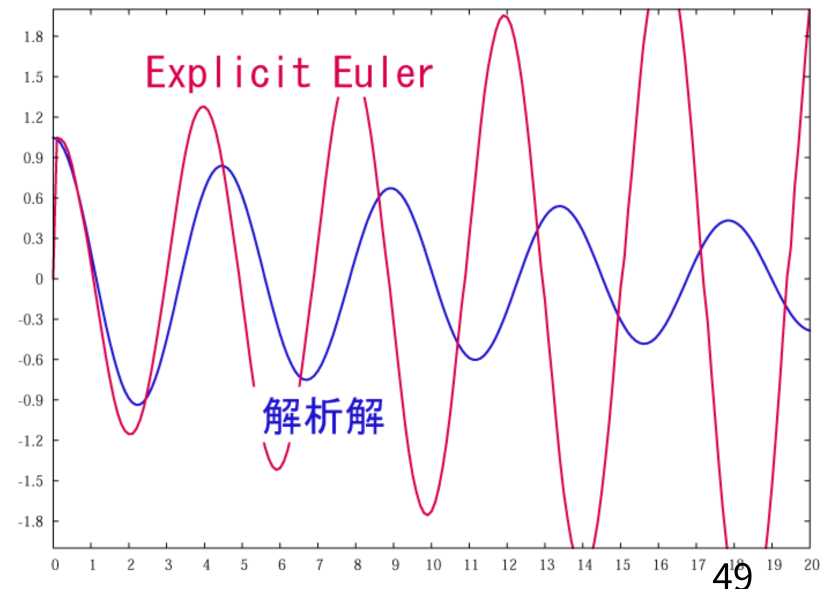
$$M\dot{v}(t) = \lambda(t)$$
$$\lambda(t) = -Kx(t) - Bv(t)$$



●Explicit Euler 法

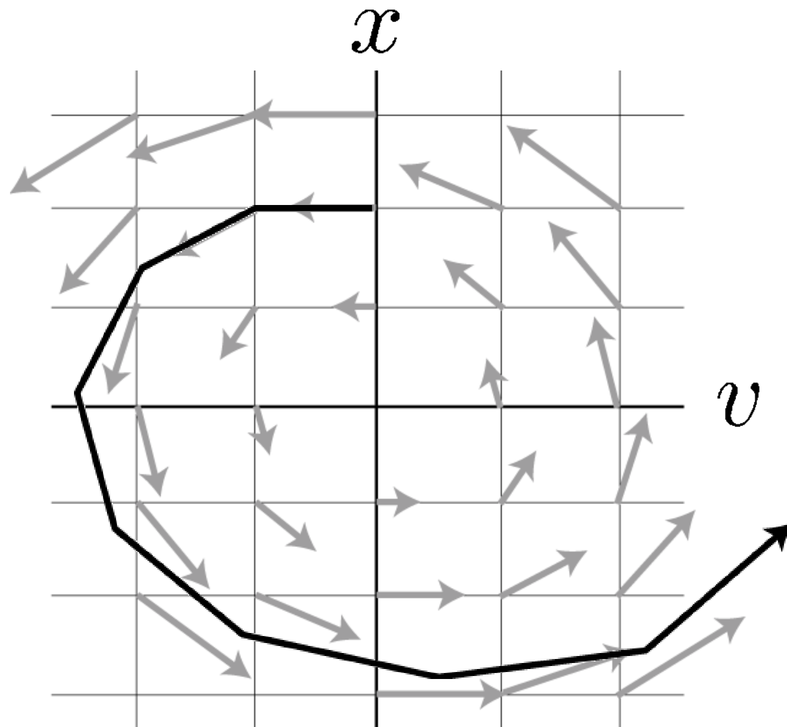
$$v[t+1] = v[t] + M^{-1}\lambda[t]\Delta t$$
$$\lambda[t] = -Kx[t] - Bv[t]$$

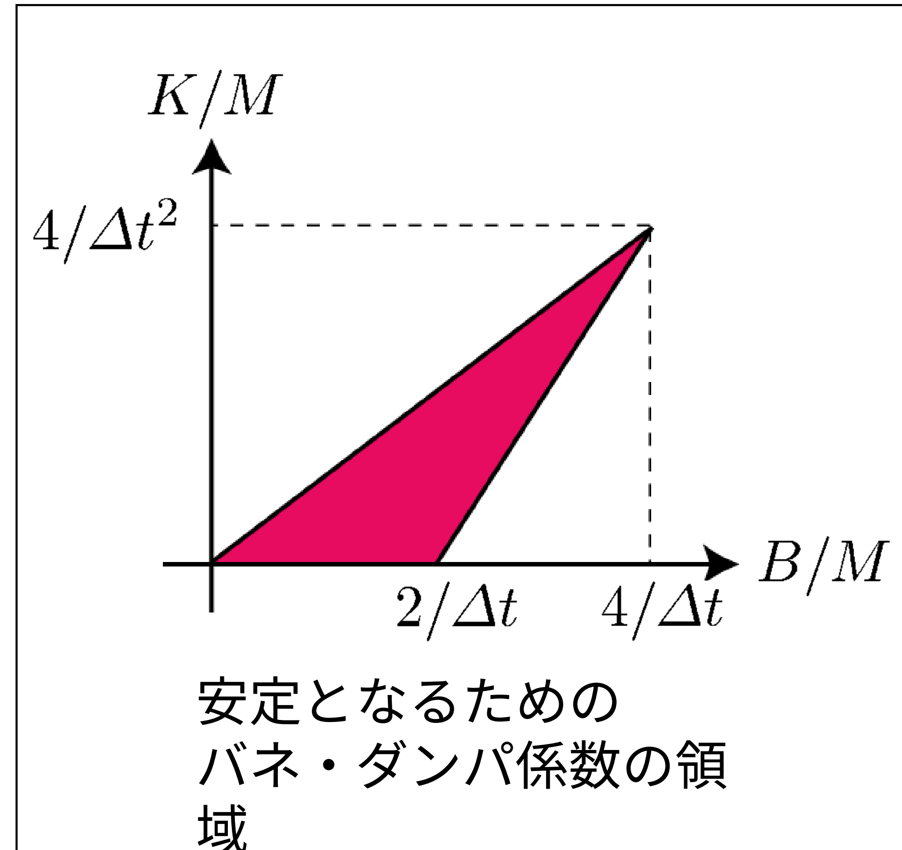
バネ・ダンパ係数やステップ幅を
大きくすると不安定となる



バネ・ダンパの計算法（**Explicit Euler法**・前進積分）

なぜ不安定？





現在時刻の位置・速度から現在時刻の力を計算

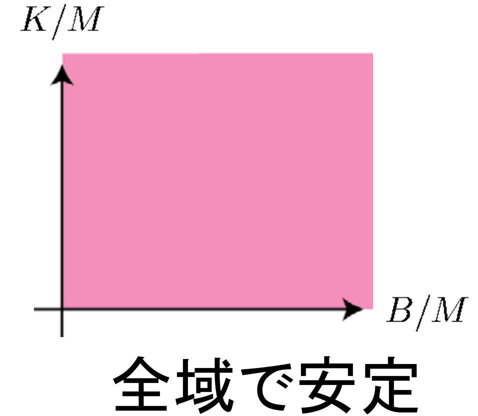
→　積分誤差による不安定化

バネ・ダンパの計算法（Implicit Euler法・後退積分）

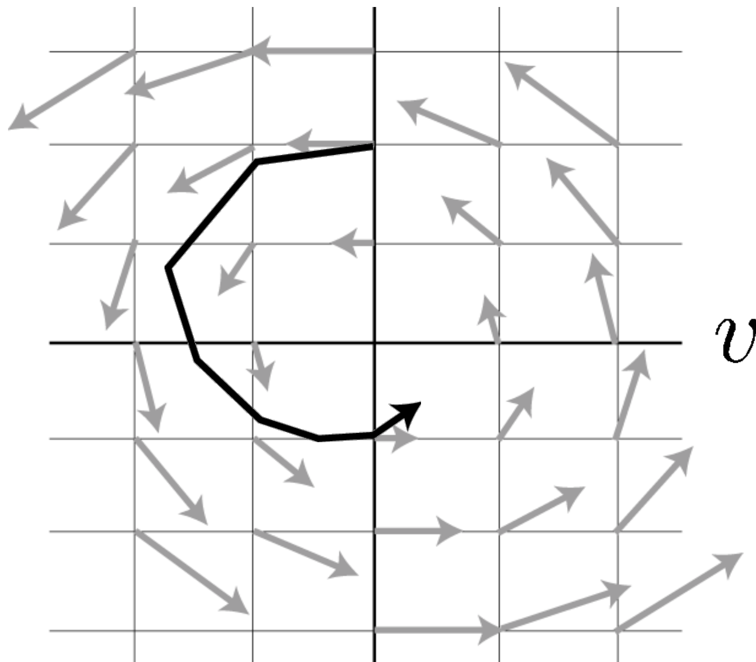
次の時刻の位置･速度から現在時刻の力を計算

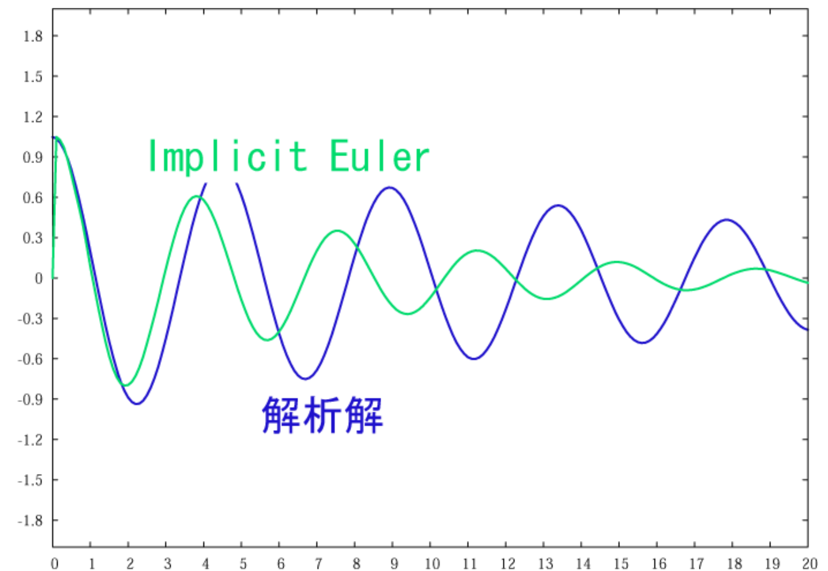
$$v[t+1] = v[t] + M^{-1}\lambda[t+1]\Delta t$$

$$\lambda[t+1] = -Kx[t+1] - Bv[t+1]$$



全域で安定





拘束を満たす時刻と安定性

加速度ベース [baraff 89など]

運動方程式 $Ma[t] = f[t] + J[t]^T \lambda[t]$

拘束式（加速度レベル） $J[t]a[t] + \dot{J}[t]v[t] = 0$

この瞬間[t]に、拘束を満たす

連立 $(JM^{-1}J^T)[t]\lambda[t] + (J[t]M^{-1}f[t] + \dot{J}[t]v[t]) = 0$

速度更新 $v[t+1] = v[t] + a[t]\Delta t$

位置更新 $p[t+1] = p[t] + v[t+1]\Delta t$

$(t \leftarrow t+1)$

積分誤差の累積によるドリフト

拘束運動の計算法

速度ベース [Stewart96など]

運動方程式 $Ma[t] = f[t] + J[t]^T \lambda[t]$

連立

$$Jv[t+1] = (JM^{-1}J^T)[t]\lambda[t]h + J[t]v[t]$$

速度更新 $v[t+1] = v[t] + a[t]\Delta t$

拘束式（速度レベル） $J[t]v[t+1] = 0$ 次の時刻[t+1]に拘束を満たす

位置更新 $p[t+1] = p[t] + v[t+1]\Delta t$

$(t \leftarrow t+1)$

積分（速度更新）を考慮して拘束力を計算→ドリフトが少ない

拘束と運動方程式の連立方程式を解く

- 拘束を代入するために、Jを使って運動方程式を変形する

$$
\begin{aligned}
M\dot{\boldsymbol{u}} &= \boldsymbol{f}_c + \boldsymbol{f}_e = \boldsymbol{J}^t\boldsymbol{\lambda} + \boldsymbol{f}_e \\
\boldsymbol{u}[t+1] &= \boldsymbol{u}[t] + M^{-1}\Delta t\boldsymbol{J}^t\boldsymbol{\lambda} + M^{-1}\Delta t\boldsymbol{f}_e \\
\boldsymbol{u}[t+1] &= \boldsymbol{u}[t] + \boldsymbol{M}^{-1}\Delta t\boldsymbol{J}^t\boldsymbol{\lambda} + \boldsymbol{M}^{-1}\Delta t\boldsymbol{f}_e \\
\boldsymbol{J}\boldsymbol{u}[t+1] &= \boldsymbol{J}\boldsymbol{u}[t] + \boldsymbol{J}\boldsymbol{M}^{-1}\Delta t\boldsymbol{J}^t\boldsymbol{\lambda} + \boldsymbol{J}\boldsymbol{M}^{-1}\Delta t\boldsymbol{f}_e \\
\boldsymbol{w}[t+1] &= \boldsymbol{w}[t] + \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^t\Delta t\boldsymbol{\lambda} + \boldsymbol{J}\boldsymbol{M}^{-1}\Delta t\boldsymbol{f}_e \\
\boldsymbol{w}[t+1] &= \boldsymbol{A}\boldsymbol{\lambda} + \boldsymbol{b}
\end{aligned}
$$

ここが、w[t+1]なので、

- 拘束条件と連立させる(蝶番の例)

拘束： $w_1, w_2, w_3, w_4, w_5 = 0$

$\lambda_6 = 0$

拘束は、w[t+1]についての拘束になっている

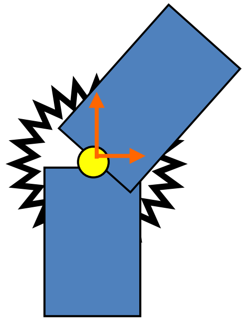
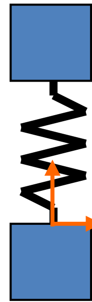
運動方程式に代入：

$$
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ w_6[t+1] \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ 0 \end{bmatrix} + \boldsymbol{b}
$$

← 連立方程式を解いて求める

Implicitのバネダンパ

●時刻[t+1]にバネダンパの制約を満たす

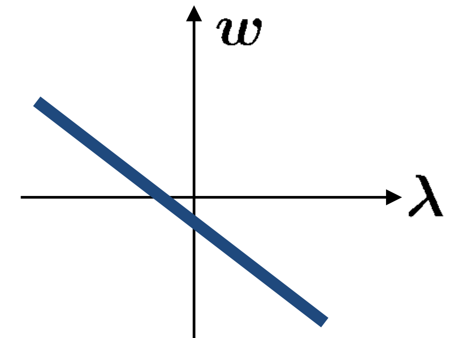
q_i	関節角度	伸び（定数）
w_i	関節角速度	伸びの速度
λ_i	関節トルク	バネダンパ力

●回転のバネダンパー

$$\lambda_6 = -K(q_6[t] + w_6[t+1]\Delta t) - Bw_6[t+1]$$

●並進のバネダンパー

$$\lambda_1 = -K(q_1 + w_1[t+1]\Delta t) - Bw_1[t+1]$$

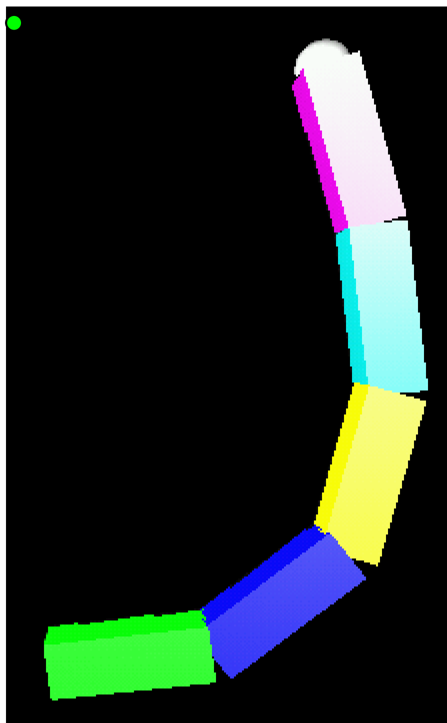


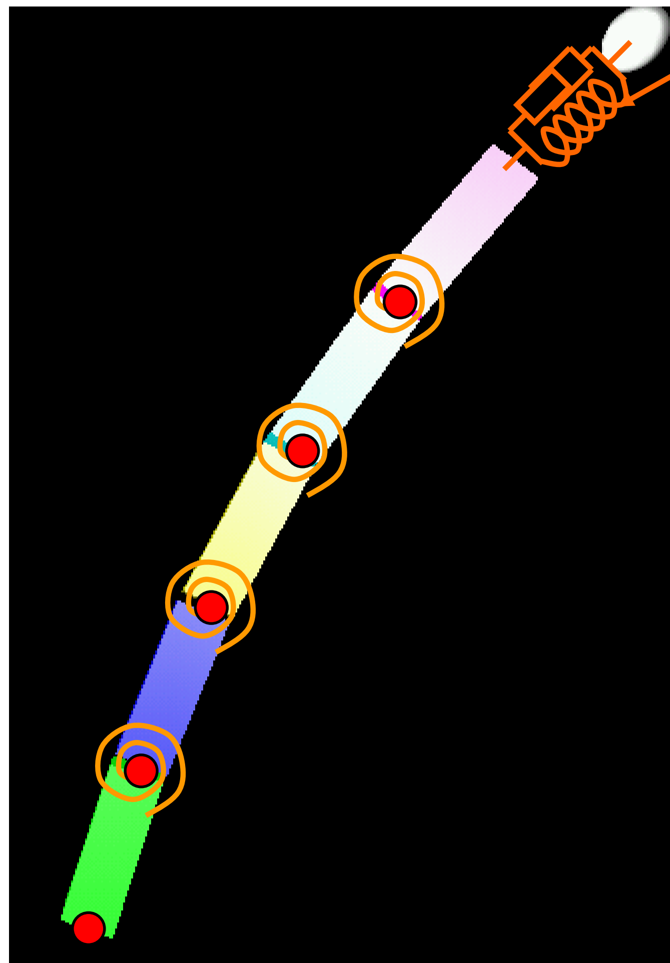
シミュレーションに組み込むことができる

安定性の比較

- 剛体
- 蝶番
- バネダンパ
- 重力

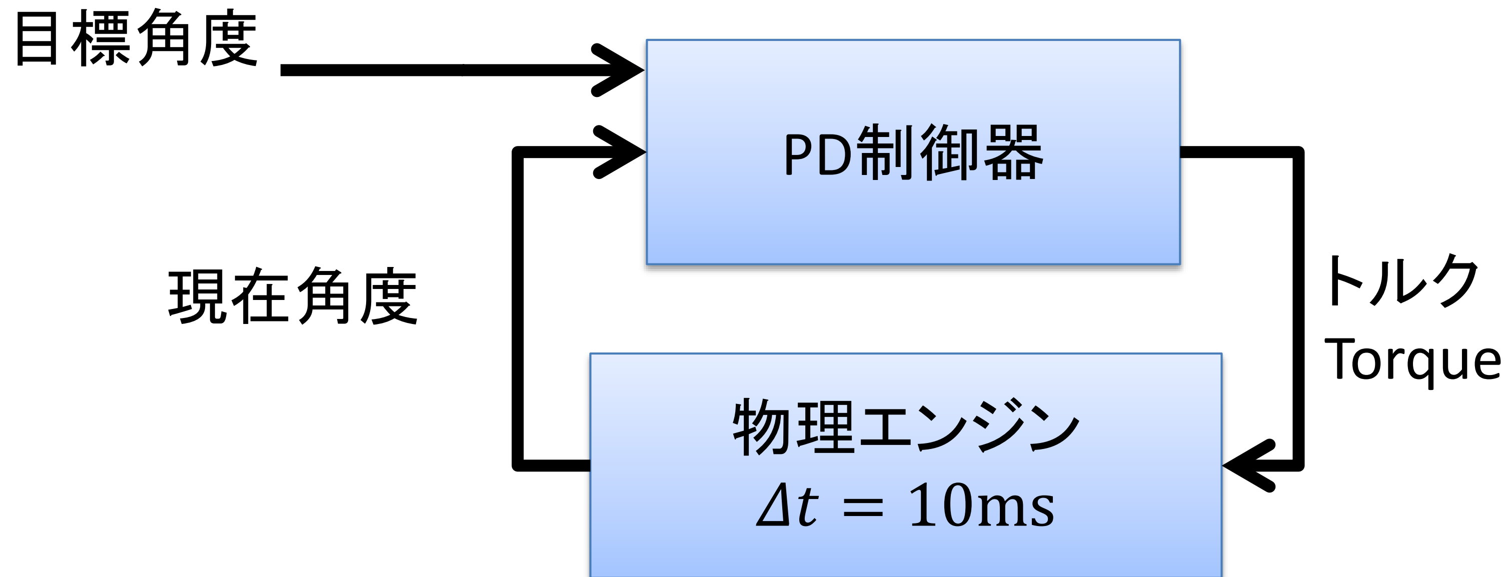
このバネ・ダンパについて
Implicit版とExplicit版を比較





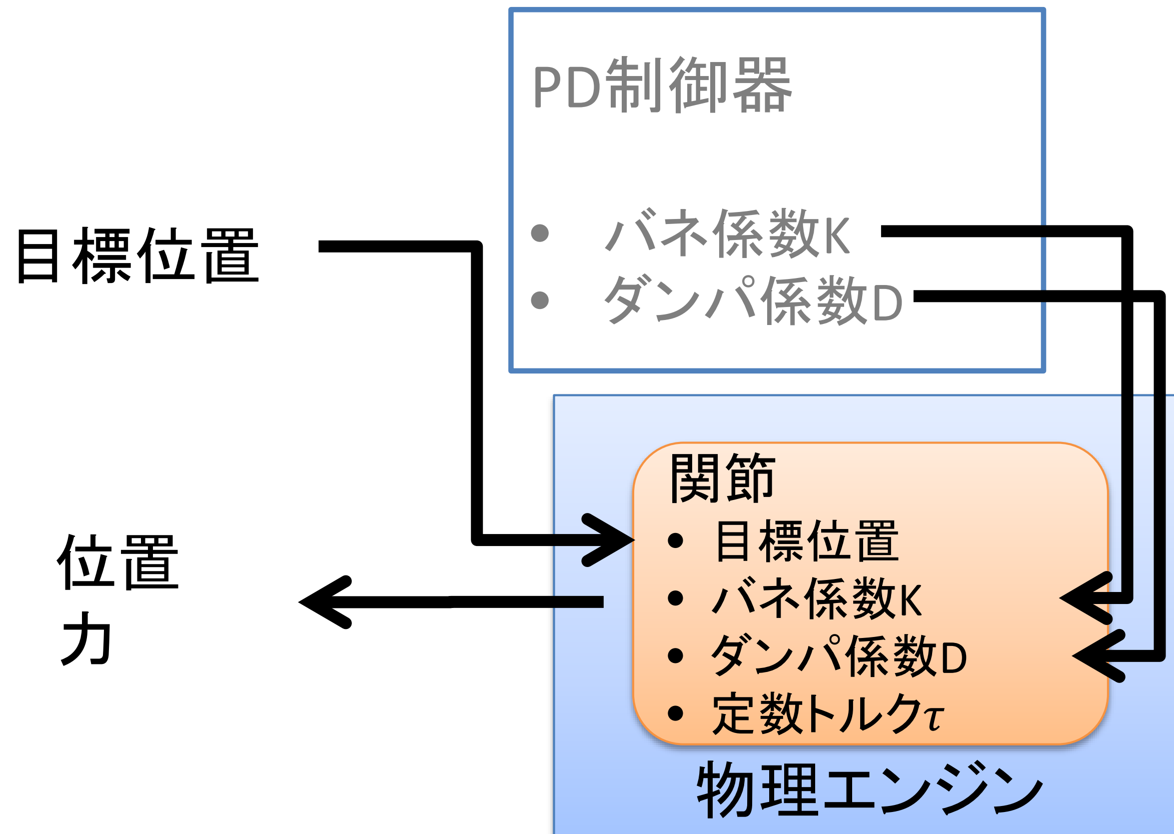
物理エンジン内のモデルの制御

- 制御器を外につけると不安定
  - Explicit（前進積分）にどうしてもなってしまう

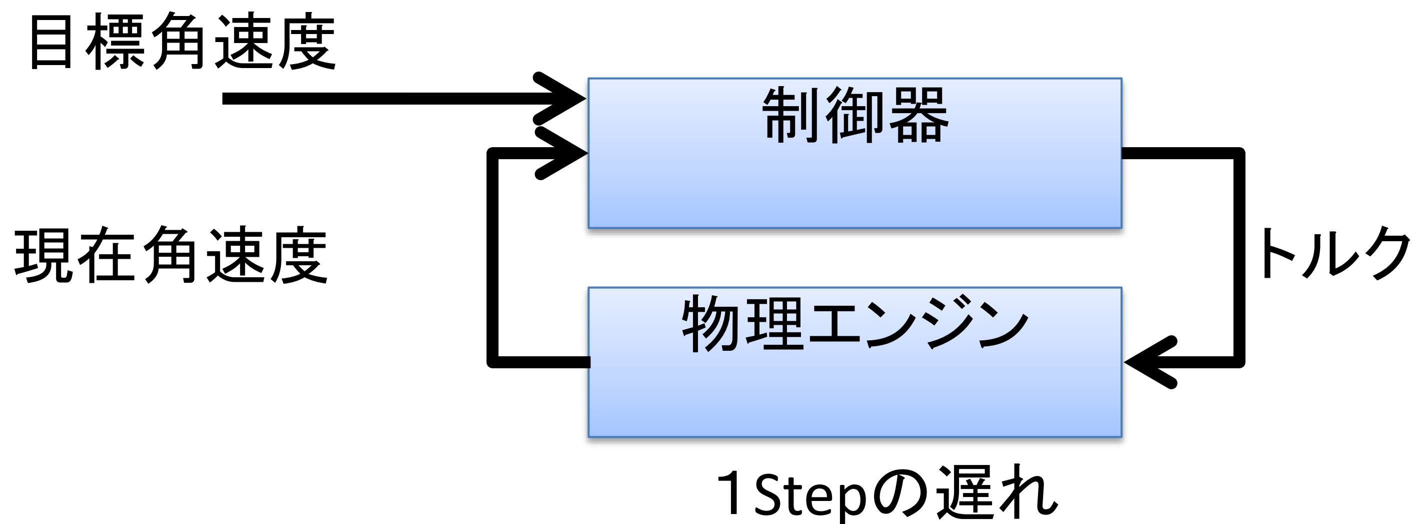


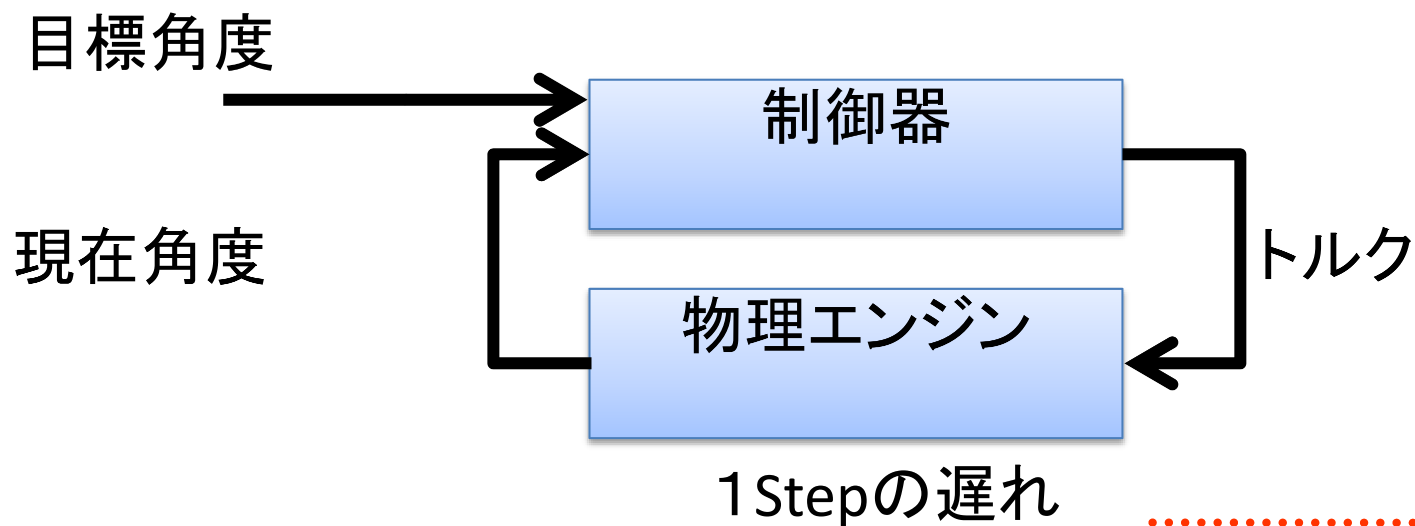
物理エンジン内のモデルの制御

- 制御器を物理エンジンの中に入れれば安定に
  - 物理エンジンの中の制御器はImplicit(後退積分)



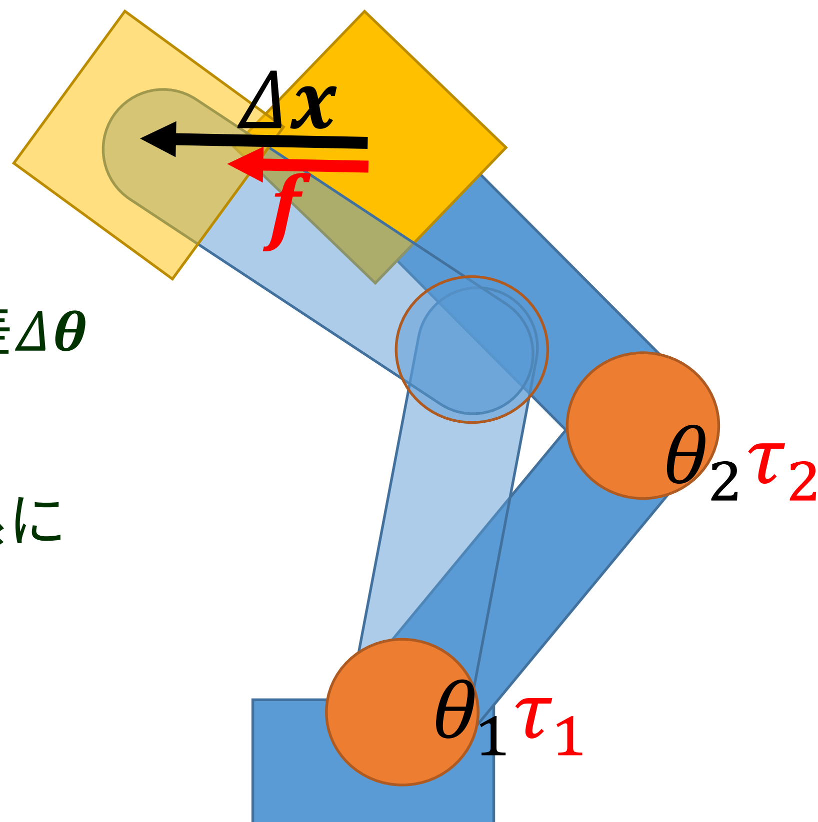
不安定になる例



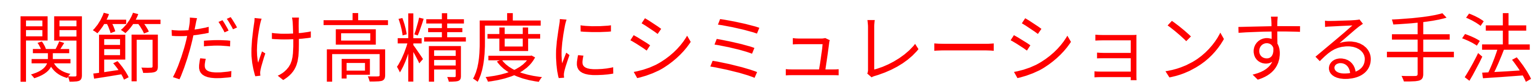


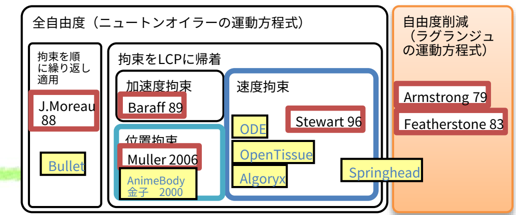
手先位置のPD制御

- 制御器が外にある場合
  手先位置の誤差$\boldsymbol{\Delta x}$ → 手先に生じる力 $\boldsymbol{f}$
  → 関節トルク$\boldsymbol{\tau}$も可能だが、
- 制御器を物理エンジンに入れたい
  手先位置の誤差$\boldsymbol{\Delta x}$ → 関節角の誤差$\boldsymbol{\Delta\theta}$
  → 目標関節角 $\boldsymbol{\theta}_{goal}$
- ヤコビアン$\boldsymbol{J}$を使って、関節座標系に
  変換すれば良い $\boldsymbol{\Delta x} = \boldsymbol{J}\boldsymbol{\Delta\theta}$



関節だけ高精度にシミュレーションする手法



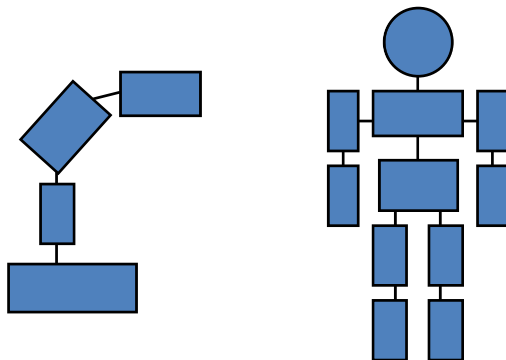


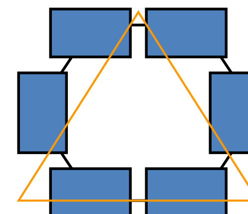
Articulated Bodyのシミュレーション

- Articulated Body
  - 多数の剛体を関節でつないだもの
  - 動物や人間の体，ロボットなど

普通にLCPでも解けるが…

- 自由度削減法
  - 可動関節だけをシミュレーション
  - 剛体が輪になっていない＝木構造 で高速な手法

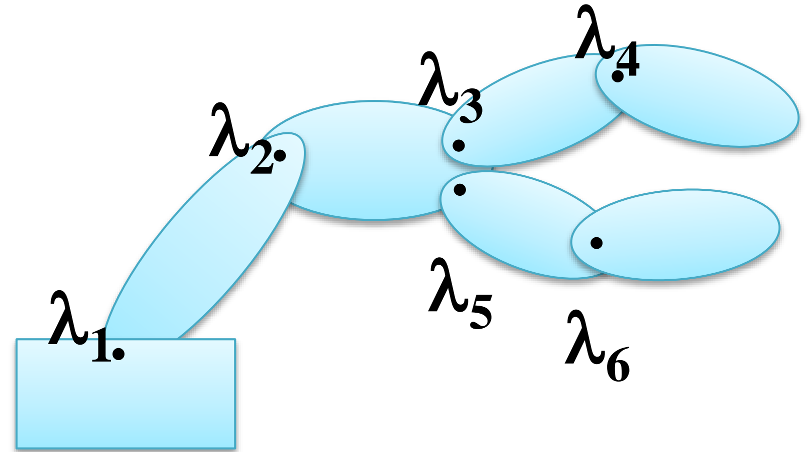




輪になっている例

繰り返し計算の誤差の問題

- 繰り返しの打ち切りにより
  剛体の速度uに誤差
  → 関節がずれる等の問題



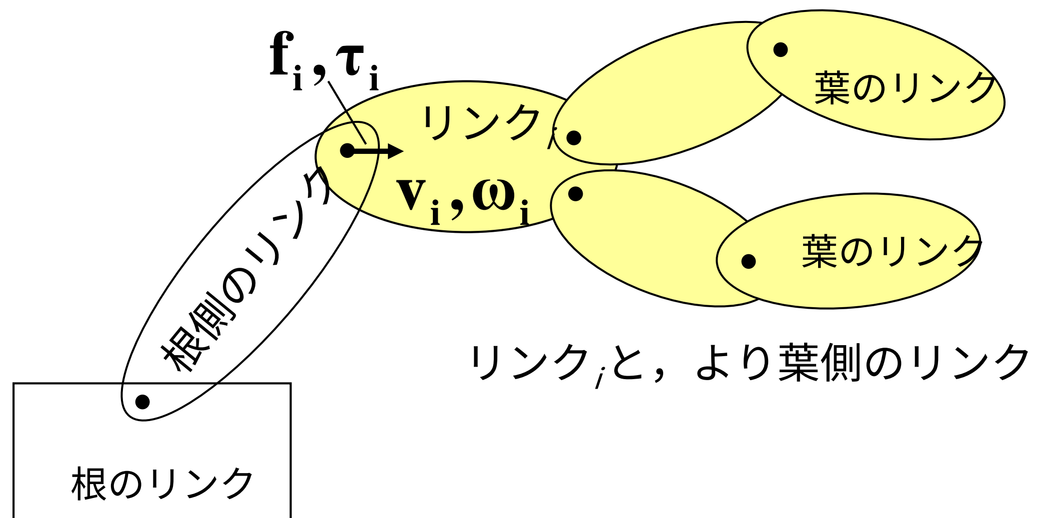
$$\boldsymbol{w}[t+1] \quad = \quad \boldsymbol{A\lambda} + \boldsymbol{b}$$

からλを解いて、剛体の速度**u**を更新
速度 **u**から剛体の姿勢**s**を更新

- 誤差をなくすには
  - 関節だけ誤差のない直接法で計算する　　(Algoryx)
  - 自由度削減法を使う (Springhead)

Featherstoneの方法

- 状態変数
  - 根のリンクの速度・角速度，位置・姿勢
  - 各関節の可動部分の角度，角速度
- リンクの位置・速度など他の値は持たない
  - 関節が外れることはない



関節の拘束力の計算

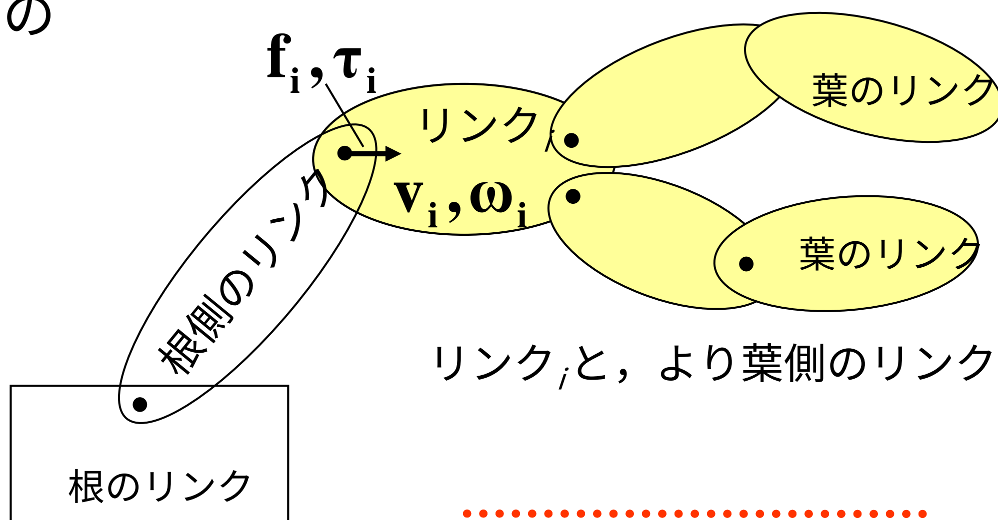
- Featherstoneの方法 木構造のリンク用
  - リンクiの運動方程式は，

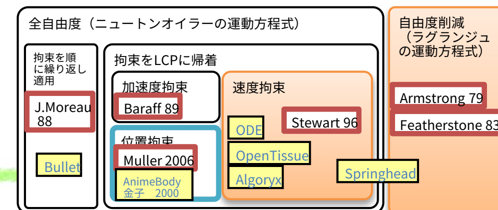
$$\begin{pmatrix} \mathbf{f}_i \\ \boldsymbol{\tau}_i \end{pmatrix} = \mathbf{I}_i \begin{pmatrix} \dot{\mathbf{v}}_i \\ \dot{\boldsymbol{\omega}}_i \end{pmatrix} + \mathbf{Z}_i$$

I_i: リンクIと葉側のリンクの慣性項
Z_i: 外力，重力，コリオリ力による項
I_i, Z_iは関節加速度によらない

のように，書ける．

  - I_i, Z_iはリンクiと葉側のリンクの姿勢・速度で決まる
  - $\boldsymbol{I_i}, \boldsymbol{Z_i}$ を葉から順に計算
  - 根側から，$\boldsymbol{v_i}, \boldsymbol{\omega_i}$を計算





FeatherstoneのLCPへの組み込み

- LCPによる定式化では、
  剛体の質量・慣性を並べて運動方程式を作った

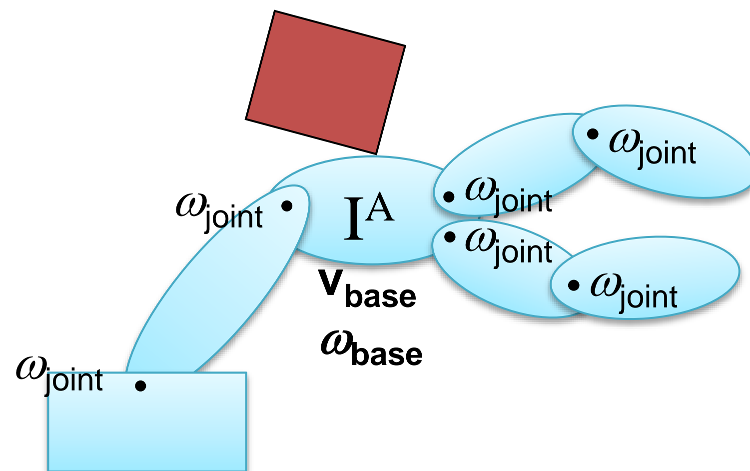
$$\boldsymbol{M}\dot{\boldsymbol{u}} = \boldsymbol{J}\boldsymbol{\lambda} + \boldsymbol{f}_e$$

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M_1} & & & \\ & \boldsymbol{M_2} & & \\ & & \ddots & \\ & & & \boldsymbol{M_n} \end{bmatrix} \quad \boldsymbol{M}_i = \begin{bmatrix} m_i \boldsymbol{E}_{33} & \\ & \boldsymbol{I}_i \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{\omega}_1 \\ \vdots \\ \boldsymbol{v}_n \\ \boldsymbol{\omega}_n \end{bmatrix} \quad \boldsymbol{f}_e = \begin{bmatrix} \boldsymbol{f}_{e1} \\ \boldsymbol{\tau}_{e1} \\ \vdots \\ \boldsymbol{f}_{en} \\ \boldsymbol{\tau}_{en} \end{bmatrix}$$

$$\boldsymbol{u}[t+1] = \boldsymbol{u}[t] + \boldsymbol{M}^{-1}\Delta t\boldsymbol{J}\boldsymbol{\lambda} + \boldsymbol{M}^{-1}\Delta t\boldsymbol{f}_e$$

- これにArticulated Bodyを加えると

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M_1} & & & \\ & \boldsymbol{I}^A & & \\ & & \ddots & \\ & & & \boldsymbol{M_n} \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{\omega}_1 \\ \boldsymbol{v}_{base} \\ \boldsymbol{\omega}_{base} \\ \vdots \\ \boldsymbol{v}_n \\ \boldsymbol{\omega}_n \end{bmatrix}$$



参考：S. Redon et al. "Adaptive Dynamics of Articulated Bodies" SIGGRAPH 2005

まとめ

- 物理エンジンと物理シミュレーションの違い
  - それまでの物理シミュレーションでは運動方程式は人が立てた
  - 物理エンジンは運動方程式を立てるのも自動
- マルチボディダイナミクスのエンジンの分類
  - 問題の種類、運動方程式、拘束の定式化、積分の仕方、方程式の解き方
- 歴史
  - 1980年　マルチボディダイナミクスの解析ツールができはじめる
  - 2000年　物理エンジン: ゲーム向けの高速, ロボット向けの高速高精度
- 物理エンジンの仕組み
  - 速度拘束をLCPに帰着させて解くタイプの物理エンジンの計算
    - 運動方程式を拘束座標系に変換して、拘束条件を代入
  - 接触判定：ブロードフェーズはソートと検索、ナローフェイズは凸だと楽
- 安定性と制御
  - Explicit(前進)とImplicit(後退)積分、Explicitは不安定
  - 制御を物理エンジンにさせれば、Implicit(後退)積分になるので安定に
- 高精度な関節
  - 繰り返し計算の打ち切りが問題、直接法やFeatherstoneの方法を使う