解　説

動作行動開発のための物理エンジン Springhead

Springhead: A physics engine for motion and behavior

長谷川　晶　一[*1]　三　武　裕　玄[*1]　田　崎　勇　一[*2]　[*1]東京工業大学　[*2]名古屋大学
Shoichi Hasegawa[*1], Hironori Mitake[*1] *and* Yuichi Tazaki[*2]　[*1]Tokyo Institute of Technology　[*2]Nagoya University

1. はじめに

ロボットを実用化するためには，様々な環境で所望の機能を果たす動作・行動を作る必要がある．そのような動作・行動の作成は，具体的作業としてはソフトウェアの開発になる．このようなソフトウェアのテストには，ロボット＋動作環境か，そのシミュレータが必要であり，テストができなければ開発を進めることもできない．実機に比べてコストが低く魅力的なシミュレータだが，その開発にも手間・コストがかかる．

近年デジタルゲーム等に多用される物理エンジンを利用することで，シミュレータ開発の手間を激減させることが期待される．物理エンジンは，手作業で求めた機構の運動方程式の数値計算等で行う通常のシミュレーションとは計算方法が異なり，特有の誤差や不安定性を持つ．このためロボットの動作・行動のテストに汎用物理エンジンを用いる場合には，特別な注意が必要になる．本稿では，物理エンジンの仕組みと特徴を説明し，ロボットの動作・行動の開発に必要なテスト環境を構築する場合の注意点を指摘する．更に，筆者らが開発を進めている，動作行動の生成とテストのための物理エンジン Springhead と動作行動開発事例を紹介する．

2. 物理エンジンとは何か

制御やその評価を目的にロボットの動作をシミュレーションする場合，まずラグランジュの運動方程式などその機構の運動を表す運動方程式を立て，次に運動方程式を数値シミュレーションするという手順がしばしば取られる．これに対して物理エンジンでは，関節等の接続情報にもとづいて運動方程式をその都度物理エンジンが自動的に立て直してシミュレーションする．このため，関節の追加・削除や

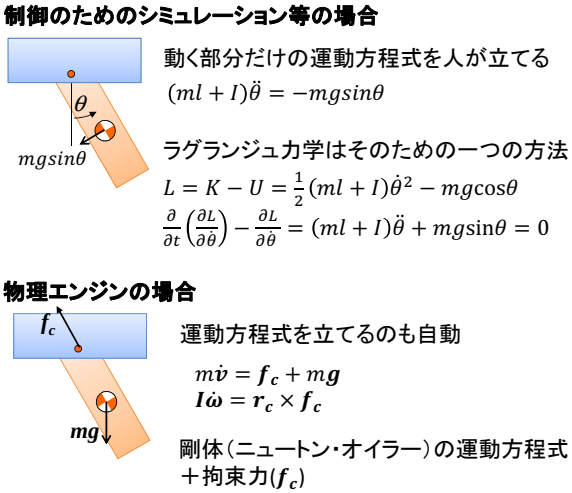


図 1　物理エンジンは運動方程式を立てる

接触状態の変化のような，運動方程式が変化する場合でも，そのままシミュレーションを続けることができる．

物理エンジンでは，運動方程式を自動的に立てる必要があるため，この手間＝計算量も考える必要がある．このため，ラグランジュの運動方程式のような一般座標系の運動方程式でなく，1 剛体の運動方程式であるニュートン・オイラーの運動方程式を用い，接触等はこの運動方程式に拘束力を加えると考えることが多い．

例えば，図 1 上のラグランジュ力学による運動方程式では，スカラー θ だけが状態変数となっており，数値積分の計算量が少ない．しかし，振子の支点が外れる場合などには運動方程式を立て直さなければならない．一方，図 1 下の剛体（ニュートン・オイラー）の運動方程式では，状態変数は 3 次元のベクトル $\boldsymbol{v}, \boldsymbol{\omega}$ であり数値積分の計算量は多い．しかし，拘束が拘束力 $\boldsymbol{f_c}$ の形で記述されているので，支点が外れる場合は拘束力計算を止めて $\boldsymbol{f_c} = \boldsymbol{0}$ とするだけで良い．

2.1 物理エンジンの処理の流れ

物理エンジンでは，接触状態や関節での拘束の変化といった拘束条件の変化を反映させるための処理を毎ステップ行う必要がある．具体的には次のような計算をシミュレーショ

原稿受付
キーワード：physics engine
[*1]神奈川県横浜市緑区長津田町 4259
[*2]愛知県名古屋市千種区不老町
[*1]4259 Nagatsuta-cho, Midori-ku, Yokohama, Kanagawa, Japan
[*2]Furo-cho, Chikusa-ku, Nagoya, Aichi, Japan

ンのステップ毎に行う．

(1) 接触判定: 剛体の位置・速度・形状から接触位置や法線の向き等を求める．

(2) 拘束条件の更新:接触判定の結果に基づいて，接触による拘束を追加する．また，関節の可動域限界による拘束の追加も行う．

(3) 拘束力の計算:拘束条件が与える拘束力を計算する．

(4) 運動方程式に基づき速度，位置を更新する．

このため，物理エンジンは運動方程式の立て方と拘束力の計算法により分類できる．

運動方程式の立て方には，

- 全自由度法：自由運動する 6 自由度剛体の直交座標系での運動方程式 (ニュートン・オイラーの運動方程式)
- 自由度削減法：関節可動軸など実際に動く自由度についての一般座標系での運動方程式 (ラグランジュの運動方程式)

がある．全自由度法では，全自由度の位置・速度をシミュレーションする．拘束の変化によって運動方程式の数が変化しないので，接触のように頻繁に変化する拘束にも対応できる．拘束は，拘束力を求めて運動方程式に加えることで実現される．そのため拘束力の計算に誤差が生じると拘束に誤差が生じる．高速化のため近似計算を用いるため，一部の拘束力が極端に大きな場合などには大きな誤差が残り，結果として関節がずれたり本来曲がらないはずの向きに動いたりしてしまうこともある．チェビシェフリンクのような大きな拘束力が働く機構を用いる場合には注意が必要である．

一方，自由度削減法は，関節が壊れるような回転など拘束が存在する方向には自由度を持たない．このため計算誤差があっても関節がずれるなどといった拘束を破るような誤差は発生しない．また自由度の数＝運動方程式の数が少ないためシミュレーションの計算量が少ない．しかし，関節を外したり追加したりといった拘束の変化が生じた際には運動方程式の数が変化するため処理が多くなる．そのため接触のような頻繁に変化する拘束には向かない．

拘束力の計算法による物理エンジンの分類と各分類に当てはまる論文と物理エンジンを図 2 に示す．ゲーム等では拘束条件を運動方程式と連立させて解く解析法と呼ばれる方法，中でも拘束条件を速度の次元で記述する手法 [6] [9] [10] [11] が良く用いられる．この手法は，運動方程式を速度の更新式とし，更新後の速度に関する拘束条件を連立させることで，陰積分によって速度を更新する．このため時間刻みを大きくすることができる (3.2 節)．これに対して拘束条件を加速度で記述する手法 [4] [8] では陽積分になってしまう．撃力法 [1] [2] は，衝突力を衝突が起きた順番に 2 体問題として計算する．大きな撃力を正確にシミュレーションができるが，接触を順番に判定する必要がある上，継続した接

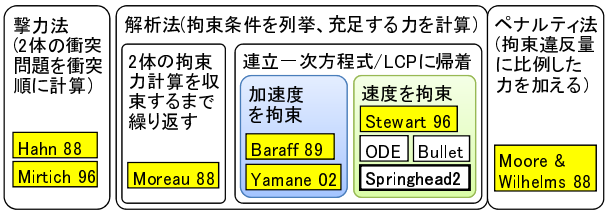


図 2 拘束の計算法による物理エンジンの分類

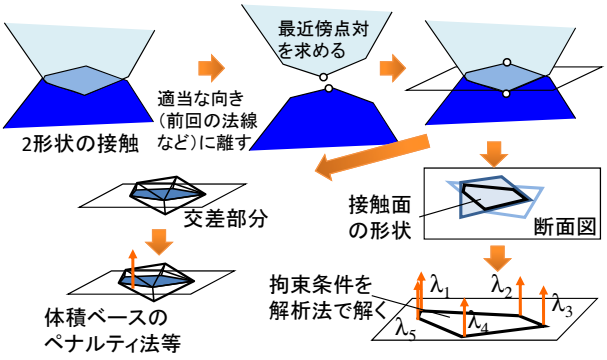


図 3 Narrow phase の接触判定の手順

触には別の手法が必要となる．一方 [3] は 2 体問題を 1 ステップの中で繰り返し計算することで多体の拘束を満たす手法で，簡単な手順で連立方程式の解を求めることができる．ペナルティ法 [7] は，拘束違反量に応じた力を与える手法で 1 ステップあたりの計算は少ないが時間刻みを小さくしなければならない．

2.2 接触判定手法

接触判定は，まず接触の可能性がある形状の対を列挙し (Broad phase)，その後 1 つの形状の対を細かく調べる (Narrow phase)．判定結果として得られる情報は Narrow phase の手法に依存する．

物体形状はポリゴンやプリミティブ等諸々の形式で表現できるが，予め凸形状や凸多角形に分解しておく．局所最小に陥る恐れがないため，距離が極小となる点の対＝最近傍点対となり [12] や [13] のような最近傍点探索アルゴリズムが知られているからである．これらのアルゴリズムは最近傍点対を求めるが，2 形状が重なってしまっていると働かない．このため，一旦形状対を侵入が解消する向きにずらしてから判定する等の工夫が必要になる．接触法線と侵入量は最近傍点対から求める事ができるが，接触面の形状や交差部分の 3 次元形状を求めるには更に計算が必要になる [14](図 3)．

2.3 拘束力計算にもとづくシミュレーション手法

本節では、多くの物理エンジンが採用している拘束力計算にもとづくシミュレーション手法を簡単に説明する。なお，本節および以降の節において，添え字 i，j を剛体や拘束を参照する場合とベクトル・行列の成分を指示する場合

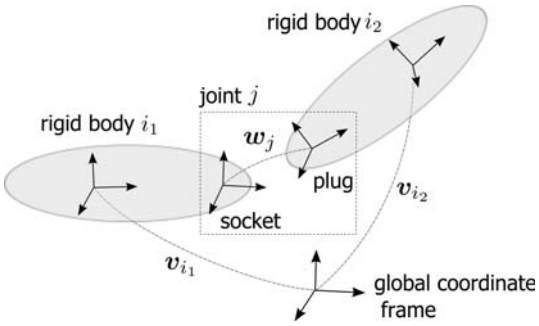


図 4 剛体と拘束の関係

の両方に用いるので注意されたい．

剛体の数を N^{B} とし，剛体 $i\,(i = 1, 2, \ldots, N^{\mathrm{B}})$ の速度を運動座標系で表現したベクトルを $\boldsymbol{v}_i$ とする．ただし $\boldsymbol{v}_i$ は並進速度の x，y，z 成分，角速度の x，y，z 成分の順に並んだ 6 次元ベクトルである．さらに $\boldsymbol{v}_i$ を全剛体について連結したベクトルを $\boldsymbol{v} \in \mathbb{R}^m$ $(m = 6N^{\mathrm{B}})$ と書く．オイラー則を適用すると，系の次時刻の速度 v' は

$$\boldsymbol{v}' = \boldsymbol{v} + h(M^{-1}(\boldsymbol{f} + \boldsymbol{f}^{\mathrm{ext}}) + \boldsymbol{c}) \qquad (1)$$

と書ける．ここで $\boldsymbol{f}$ は剛体に作用する拘束力，$\boldsymbol{f}^{\mathrm{ext}}$ は拘束力以外の値が既知である力（重力もこの中に含まれる），$\boldsymbol{c}$ はコリオリ力と遠心力に由来する加速度である．いずれも各剛体に関する並進成分と回転成分から成る $6N^{\mathrm{B}}$ 次元ベクトルで，速度と同様に剛体の運動座標系表現である．また，M は各剛体の質量と慣性モーメントから成るブロック対角行列，h はステップ幅である．一方，拘束の数を N^{J} とする．拘束は二つの剛体の組に拘束力を作用し，剛体間の相対運動に制限を加える要素である．最も典型的なものは種々の関節（回転関節，直動関節，球面関節など）であるが，接触や後述するバネ・ダンパなども拘束として扱われる．剛体 i_1，i_2 間に拘束 $j\,(j = 1, 2, \ldots, N^{\mathrm{J}})$ が作用するときの位置関係を図 4 に示す．このとき，剛体 i_1 にはソケット，剛体 i_2 にはプラグと呼ばれる座標系が取り付けられる（ソケット/プラグという呼称は Springhead に準じる）．ここでソケットから見たプラグの相対速度をソケット座標系で表現したベクトルを $\boldsymbol{w}_j$（$\boldsymbol{v}_i$ 同様並進速度と角速度の 6 次元ベクトル）とおくと，

$$\boldsymbol{w}_j = J_{i_2,j}\boldsymbol{v}_{i_2} - J_{i_1,j}\boldsymbol{v}_{i_1} \qquad (2)$$

と書ける．ただし $J_{i_1,j}$，$J_{i_2,j}$ はソケット/プラグの位置や剛体同士の位置関係から決まるヤコビ行列である．全拘束について $\boldsymbol{w}_j$ を連結したベクトルを $\boldsymbol{w} \in \mathbb{R}^n$ $(n = 6N^{\mathrm{J}})$ とおくと，上の式はまとめて

$$\boldsymbol{w} = J\,\boldsymbol{v} \qquad (3)$$

と書ける．本稿では $\boldsymbol{w}_j$ や $\boldsymbol{w}$ を拘束速度と呼ぶ．

拘束が剛体に作用する拘束力を $\boldsymbol{\lambda} \in \mathbb{R}^n$ と書く．仮想仕事の原理から $\boldsymbol{\lambda}^{\mathsf{T}}\boldsymbol{w} = \boldsymbol{\lambda}^{\mathsf{T}}J\boldsymbol{v} = \boldsymbol{f}^{\mathsf{T}}\boldsymbol{v}$ が言えるから，$\boldsymbol{f} = J^{\mathsf{T}}\boldsymbol{\lambda}$ を得る．これを式 (1) に代入すると，次時刻の拘束速度 $\boldsymbol{w}' = J\boldsymbol{v}'$ は拘束力 $\boldsymbol{\lambda}$ に関する一次式として次のように書ける．

$$\boldsymbol{w}' = A\boldsymbol{\lambda} + \boldsymbol{b}, \qquad (4)$$
$$A = h(JM^{-1}J^{\mathsf{T}}),\ \boldsymbol{b} = J(\boldsymbol{v} + h(M^{-1}\boldsymbol{f}^{\mathrm{ext}} + \boldsymbol{c})).$$

拘束速度 $\boldsymbol{w}'$ と拘束力 $\boldsymbol{\lambda}$ の各成分には，拘束の種類に応じて異なる代数条件が課せられる．これを以下の表にまとめる．

種類	表記	条件
無拘束	$i \in \mathcal{I}^{\mathrm{f}}$	$\lambda_i = 0$
基本拘束	$i \in \mathcal{I}^{\mathrm{e}}$	$w'_i = 0$
相補性拘束	$i \in \mathcal{I}^{\mathrm{c}}$	$w'_i \geq 0,\ \lambda_i \geq 0,\ w'_i\lambda_i = 0$
線形拘束	$i \in \mathcal{I}^{\mathrm{s}}$	$w'_i = \alpha_i\lambda_i + \beta_i$

(5)

ここで任意の $i \in \{1, 2, \ldots, n\}$ は $\mathcal{I}^{\mathrm{f}}$, $\mathcal{I}^{\mathrm{e}}$, $\mathcal{I}^{\mathrm{c}}$ および $\mathcal{I}^{\mathrm{s}}$ のいずれかに含まれ，かつ重複はないものとする．無拘束は文字通り拘束が課されないこと場合を示し，拘束力を生じない．自由回転する関節の軸方向の角速度成分などがこれに該当する．基本拘束はこれと逆の場合で，拘束速度が 0 となるように拘束力が生じる．相補性拘束は接触や可動範囲拘束を扱う場合に現れる．最後に線形拘束は関節にバネ・ダンパ（あるいは PD 制御器）を加える場合に現れる．線形拘束は変形により基本拘束に帰着できる．これについては 3.2 節で詳しく述べる．

例として 1 自由度の回転関節を実現したい場合は，$\boldsymbol{w}_j$ の 6 成分のうち並進 x，y，z と回転 x，y 成分に基本拘束，回転 z 成分には無拘束（自由回転）あるいは線形拘束（アクチュエータあり）を課せばよい．このとき z 軸が回転軸となる．また，接触を実現するには，まず接触点において各剛体にソケットとプラグを x 軸が接触法線を向くように取り付け，並進 x 成分に相補性拘束，y，z 成分に基本拘束，回転 x，y，z 成分を無拘束とする．ただしこの場合は静止摩擦係数は ∞ となる（並進 y，z 成分を無拘束とすれば摩擦係数 0 となる）．より一般の静止摩擦係数や動摩擦を表現するには並進 y，z 成分に関して相補性拘束に似た条件を課すことになる．

以上より，拘束力を求める問題は式 (4)(5) で与えられる線形相補性問題 (linear complimentarity problem, LCP) となる．物理シミュレーションにおける LCP の解法としては当初 Lemke 法が用いられていたが [5]，後により計算コストの低い反復解法が広く用いられるようになった [15]．Springhead が採用しているガウス-ザイデル法に基づくアルゴリズムを **Algorithm** 1 に示す．前述のように線形拘束は基本拘束に帰着できるので本アルゴリズムでは直接扱ってい

Algorithm 1 ガウス・ザイデル法による拘束力計算

```
λ ← 0, w' ← b
for k = 1 to K do
    // 1. constraint force update
    for i = 1 to n do
        if i ∈ I^f then
            continue
        end if
        λ'_i ← λ_i − A_ii^{-1} w'_i
        if i ∈ I^c and λ'_i < 0 then
            λ'_i ← 0
        end if
        δλ_i ← λ'_i − λ_i
        // 2. velocity update
        for j = 1 to m do
            δv_j ← (M^{-1} J^T)_ji δλ_i, v_j ← v_j + δv_j
            // 3. constraint error update
            for i = 1 to n do
                w'_i ← w'_i + J_ij δv_j
            end for
        end for
    end for
end for
return λ
```

ない．各反復において，拘束の各成分を巡回し，$i \in \mathcal{I}^{\mathrm{e}} \cup \mathcal{I}^{\mathrm{c}}$ であれば w'_i が 0 となるように λ_i を修正するという手続きを行う．加えて，相補性拘束（$i \in \mathcal{I}^{\mathrm{c}}$）であれば更新後に $\lambda_i \geq 0$ となるように射影操作を施す．ガウス・ザイデル法の反復回数 K はユーザが設定する．当然 K を大きく設定するほど計算精度は向上するが，計算時間は比例的に増大する．

2.4 関節座標系にもとづくシミュレーション手法

前節で述べた LCP にもとづく方法は，閉リンク構造や接触等の動的に生成される拘束を容易に扱えるという利点がある一方で，反復回数が少ないと残留誤差が生じるという難点がある．これとは対照的に，以下で紹介する関節座標上のシミュレーション技法は開リンク構造に限定されるものの関節拘束が厳密に満たされるという利点を持つ．中でも Featherstone によって発案された ABA(articulated body algorithm) は，木構造を成すリンク系の順動力学計算を O(N) の計算量（N はリンク数）で行うことができる [16].

木構造のある節に対応する剛体 i に注目し，剛体 i およびその下に連なる剛体から成る部分木を考える．このとき，部分木の根である剛体 i に作用する外力 $\boldsymbol{f}_i$ と加速度 $\boldsymbol{a}_i$ には以下の線形な関係が成り立つ．

$$\boldsymbol{f}_i = I_i^{\mathrm{A}} \boldsymbol{a}_i + \boldsymbol{z}_i^{\mathrm{A}} \tag{6}$$

ここで I_i^{A} は，剛体 i およびそれに連なるすべての剛体の慣性を反映した行列で，articulated inertia matrix と呼ばれる．また，$\boldsymbol{z}_i^{\mathrm{A}}$ は外力と拮抗して剛体 i の加速度を 0 とする力であるので，articulated bias force と呼ばれる．この二つの変数を再帰的に計算するのが ABA の基本的な考えである．以下に ABA の処理の大枠を示す．

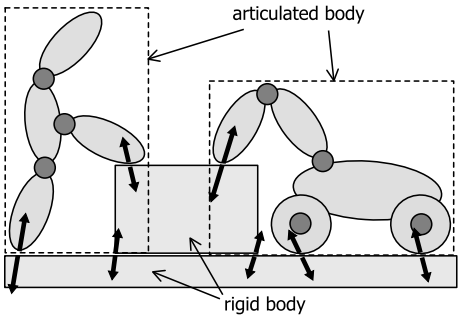


図 5 拘束力計算の枠組みへの関節座標系の統合

(1) 木の末端から根に向かい，I_i^{A}, $\boldsymbol{z}_i^{\mathrm{A}}$ を再帰的に計算
(2) 根剛体の加速度を決定
(3) 根から末端に向かって加速度 $\boldsymbol{a}_i$ と関節加速度 $\ddot{q}_i$ を再帰的に計算

より具体的な解説については文献 [16] を参照されたい．本稿では 3.1 節において ABA を LCP の枠組に組み入れる方法について解説する．

3. 動作行動生成のための物理エンジン

ロボットの動作や行動の生成では，関節を持つロボットのシミュレーションが多くなる．また，機構だけでなく制御を含めたシミュレーションが重要になる．本節では，筆者らが開発している物理エンジン Springhead [11] が持つ，動作行動生成に便利な機能を紹介する．

3.1 拘束力計算法と関節座標表現の統合

2.3 節で述べた拘束力計算による手法と 2.4 節で述べた関節座標表現による方法は互いに相補的な特性を持つといえる．そこで両者をひとつのシミュレーション環境で混在させることを考える．LCP に基づく拘束力計算法では系の構成要素は必ずしも剛体である必要は無く，拘束力に対して速度変化の計算を効率的に行える任意のサブシステムを組み入れることができる．この性質を利用するために，図 5 に示すように系全体をいくつかのサブシステムに分割する．ここで一つのサブシステムは単一の剛体か，ループを持たないツリー状リンク構造 (articulated body) とする．このようなグループ分割を行うと，サブシステム内部に生じる内力の計算は行わずに，サブシステム間に作用する拘束力の計算のみで済むようになる．その代わりに各サブシステムについて，外部から加わった拘束力に対するそのサブシステムに属する剛体の速度変化を計算する必要が生じるが，これは 2.4 節で紹介した ABA を応用することで効率的に行うことができる．

3.2 高ゲイン PD 制御器のシミュレーション技術

ロボットの物理シミュレーションにおいては，関節のサーボ制御のために高ゲインの PD コントローラを用いることが多い．しかしながら，数値積分にオイラー則を用いる場

合に安定性を保証するにはゲイン係数に応じて十分小さなステップ幅を設定する必要があり，計算時間に制約のあるリアルタイムアプリケーションでは設定可能なゲイン係数に限界がある．そこで，以下では陰積分の考えにもとづき PD 制御やバネ・ダンパ系をある種の拘束と見なすことで LCP の枠組みに取り入れ，比較的粗いステップ幅のもとで安定なシミュレーションを実現する方法を述べる．

2.3 節で述べた枠組みの中で，i 番目の拘束自由度に対して以下のようにして PD 制御を行うことを考える．

$$\lambda_i = K_i(\theta_i^{\mathrm{d}} - \theta_i) + D_i(w_i^{\mathrm{d}} - w_i) + \tau_i \qquad (7)$$

ここで K_i，D_i は制御ゲイン（あるいはバネ・ダンパ係数），θ_i は現時刻での関節変位，θ_i^{d} は目標変位，w_i^{d} は目標速度，τ_i は定数項である．例えば $K_i > 0$，$D_i > 0$，$w_i^{\mathrm{d}} = 0$ とすれば変位に関する PD 制御となり，$K_i = 0$，$D_i > 0$ とすれば速度に関する比例制御となる．定数項 τ_i は必要に応じて用いる．式 (7) の方法ではどうしても 1 ステップの遅れが生じるため，決められたステップ幅 h のもとではゲイン係数をある程度以上大きく設定できない．そこで陰積分の考えにもとづき，上の式中の関節の速度と変位を次時刻のものに置き換える:

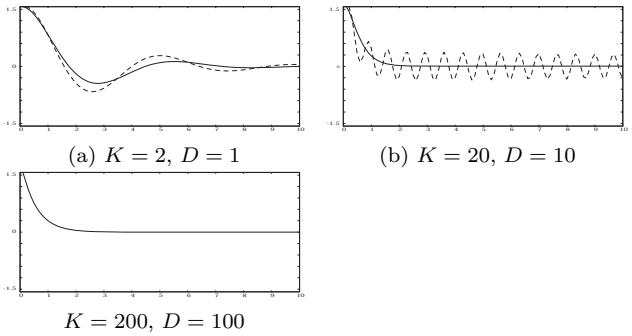
$$\begin{aligned}\lambda_i &= K_i(\theta_i^{\mathrm{d}} - (\theta_i + hw_i')) + D_i(w_i^{\mathrm{d}} - w_i') + \tau_i,\\ w_i' &= -\frac{1}{D_i + hK_i}\lambda_i + \frac{K_i(\theta_i^{\mathrm{d}} - \theta_i) + Dw_i^{\mathrm{d}} + \tau_i}{D_i + hK_i}\end{aligned} \qquad (8)$$

ただし次時刻の関節変位は $\theta_i + hw_i'$ と近似している．式 (8) は 2.3 節で述べた線形拘束に該当する．ここで，式 (8) と式 (4) を連立し整理すると，あらたに $\tilde{A}$，$\tilde{\boldsymbol{b}}$ を

$$\begin{aligned}\tilde{A}_{i,j} &= \begin{cases} A_{i,j} + \frac{1}{D_i + hK_i} & \text{if } i = j \text{ and } i \in \mathcal{I}^{\mathrm{s}} \\ A_{i,j} & \text{otherwise} \end{cases}\\ \tilde{b}_i &= \begin{cases} b_i - \frac{K_i(\theta_i^{\mathrm{d}} - \theta_i) + Dw_i^{\mathrm{d}} + \tau_i}{D_i + hK_i} & \text{if } i \in \mathcal{I}^{\mathrm{s}} \\ b_i & \text{otherwise} \end{cases}\end{aligned} \qquad (9)$$

と定義すると，関係式 $\boldsymbol{w}' = \tilde{A}\boldsymbol{\lambda} + \tilde{\boldsymbol{b}}$ のもとで線形拘束は基本拘束に帰着される．

式 (7) の従来手法と陰積分法に関して，いくつかの異なる係数のバネ・ダンパ系の時間応答を比較する．支点に線形バネ・ダンパを取り付けた単振り子に対し，振り子の角度（=バネの変位）90 °を初期値としたときの時系列を図 6 に示す．横軸は時刻 [s]，縦軸は振り子の角度 [rad] である．振り子の長さを 0.5[m]，振り子先端の質量を 1.0[kg]，積分幅を 0.1[s] とした．陰積分法による時系列は実線，従来手法によるものは破線で描かれている．バネ係数 $K = 2.0$，ダンパ係数 $D = 1.0$ の場合（上段）ではいずれの手法においても安定性は保たれている．$K = 20.0$, $D = 10.0$ の場合（中段）では従来手法の時系列に定常的な発振が見られ，従来手法の限界に達していることが観察される．下段は $K = 200.0$, $D = 100.0$ の場合で陰積分法の時系列のみを描いたものである．このような大きなバネ・ダンパ係数を設定しても安定性が保たれることが分かる．ただし，陰的積分則はあくまで数値積分の安定性を保証するものであり，必ずしも精度が向上するものではないことは注意しなければならない．



(a) $K = 2$, $D = 1$　(b) $K = 20$, $D = 10$

$K = 200$, $D = 100$

図 6　通常のオイラー則と陰的オイラー則における応答の比較

3.3　フィードフォワードのためのトルク計算

目標軌道を追従する場合，誤差や遅れをなくすには単純な PD 制御では高ゲインが必要であり，ゲインを低くするにはフィードフォワードが必要になる．機構の運動方程式からフィードフォワードに必要なトルクを計算できるが，運動方程式を立てなければならないのは面倒である．本節では，物理エンジンを用いてトルクを求める手法 [17] を紹介する．

状態変数を保存・再現する機能を持つ物理エンジンでは，世界の状態を保存しておき必要に応じて戻すことができる．これを利用して以下のようにフィードフォワードに必要なトルクを計算しながらシミュレーションを進める．

(1) 現在の状態を保存する．
(2) 制御対象の関節の可動自由度に目標軌道の速度の速度拘束を追加する．可動自由度は無拘束となっているので，ここに $w_i = w_t$(w_t は目標軌道の速度) の拘束を加える．
(3) 接触力が働かないよう設定して，シミュレーションを１ステップ進め，拘束力を求める．
(4) (2) で追加した拘束を元に戻し，(1) の状態に物理エンジンを戻す．
(5) (3) で求めたトルクを関節に加える．
(6) 関節にインピーダンスを与えるために，軌道を目標とする低ゲインの PD 制御を 3.2 節の手法で行う．
(7) シミュレーションを進める．

各関節に (3) のトルクを与えるので (6) の PD 制御を加えなくても軌道を追従する．外乱への応答は (6) の PD 制御に依る．

4. 動作生成と開発環境

ロボットの動作は，環境とインタラクションをしながら目的を果たすように設計する必要がある．そのためテストでは環境を含めて手早く構築・調整を行いシミュレーションできることが求められる．また，動作や行動の調整には多種多様なパラメータの調整が必要な事が多く，調整とテストを繰り返す必要がある．更に，不具合を見つけた場合にその原因の追求を容易にするためには，動作や行動の原因や生成過程を開発者が把握しやすいと良い．

上記のような要求を満たすため，我々は，物理エンジンをアニメーション生成のためのモデリングツールである Blender に組み込んだ開発環境 SprPyBlender(図 7) を開発した．本節では，SprPyBlender の構成とその特徴，利用法，GUI の拡張方法を紹介する．

4.1 SprPyBlender の構成と特徴

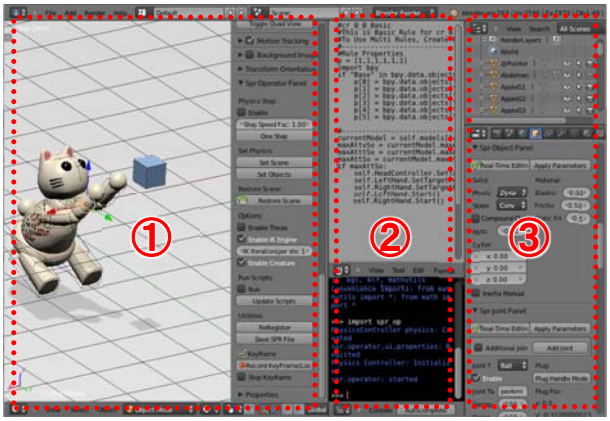


図 7 SprPyBlender
(1)3D 編集画面 (2)Python コンソール (3) パラメータ等の調整パネル

Blender は，３次元コンピュータグラフィクスによるアニメーション動画を制作することを目的に開発されているオープンソースのツールで，ゲームの制作でもモデリング等にしばしば用いられる．３次元形状を作成し，関節で繋いだり，配置したりすることで３次元のシーンを構築する作業に向いている．Blender の GUI は Python というスクリプト言語で記述されており，スクリプトを追加することで容易に GUI を始めとした機能を拡張することができる．

SprPyBlender では，まず C++言語で記述された物理エンジンである Springhead を Python で利用できるように Python ラッパーを作成し，Blender 拡張用の Python スクリプトとして物理エンジンを Blender に組み込む (図 8)．これにより，Blender 本体には手を加えずに，拡張用 Python スクリプトとして SprPyBlender のスクリプトとバイナリを追加するだけで，Blender に物理エンジンと GUI を追加している．このため Blender 本体の更新の影響を受けにくい．また Python はスクリプト言語であるため，即座に実行したり実行中のスクリプトを書き換えることもでき，効率良くテストを進めることができる．物理エンジン自体も

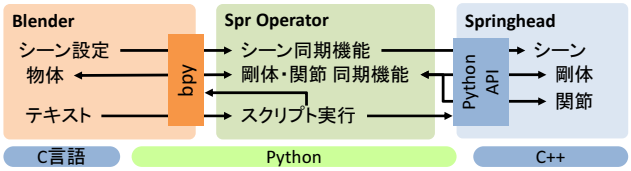


図 8 SprPyBlender の構成

Python で操作できるので，ロボットの動作指令や行動ルールも Python で記述することができる．また，Python から GUI や３次元モデルやシーンを容易に操作できるため，パラメータ調整のための GUI や可視化のための３次元グラフィクスも Python により容易に作成できる．

4.2 SprPyBlender の利用例

動作設計に利用するには，まず３次元形状を部品ごとに作成し，関節で繋いで物理モデルを構成する．形状の編集や関節位置の指定は Blender の GUI を用いて行う (図 9)．作成した形状は衝突判定に使用される．また，部品ごとの質量や関節のバネ・ダンパ係数等の指定はスライダや数値入力で行う (図 10)．動作中に数値を変更すると動作に即時反映されるため，様子を確認しながら調整を進めることができる． 動作指令や行動ルールは，Python スクリプトの

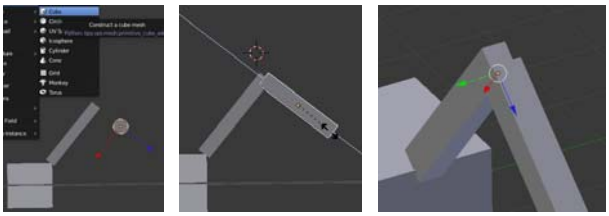


図 9 GUI を用いた物理モデルの作成

図 10 関節のバネ係数を調整・即時反映した例

形でテキスト入力する．スクリプトにはセンサ入力に対応する動作指令の生成法を記述しておき，シミュレーションの１ステップごとに実行される．

4.3 GUI の拡張例と可視化

センサ特性などロボット固有の値に対して，新たに調整 UI を追加することもできる．数値を入力する UI の追加は，まず数値プロパティと呼ばれるデータオブジェクトを用意し，これを Blender UI に登録することで行う．数値プロパティに保持されたデータはシーンをファイルにセーブ・ロードすると自動的に保存・復元される．行動ルールを記述した Python スクリプトから，対応する数値プロパティの値を参照することで動作への反映がなされる．

設定した数値を可視化する事でより直感的な調整作業も可能となる．図 11 では円錐形状で視野角の可視化を行なっている．Blender 上の形状の作成・編集は bpy と呼ばれる Python インタフェースを用いてスクリプトから行うことができる．詳しくは bpy リファレンス [18] を参照されたい．

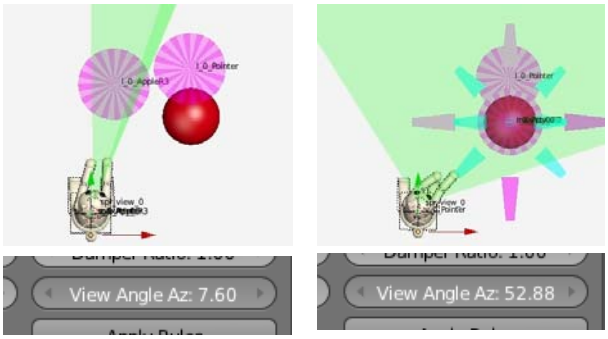


図 11 視野角調整用の UI

5. お わ り に

動作行動の開発はロボットの実用化に欠かせないものであり，今後ますます重要になると考えられる．機能の高度化につれて，動作行動開発が高度で複雑なソフトウェア開発となることは避けられないだろう．

ソフトウェア開発ではテストを繰り返すので，変更してからテストするまでに要する時間（開発のターンアラウンドタイム）を短く楽にすることが重要である．物理エンジンを用いたシミュレーション環境は，今後も効率の良い開発に欠かせない技術として発展していくであろう．また，前述の通り，動作行動の開発にはロボット自体だけでなくロボットが活動する環境もシミュレーションする必要があり，環境の構築や変更を手早く行えることも重要である．そのため，本稿では筆者らが構築した GUI を用いたシーン作成環境を紹介した．

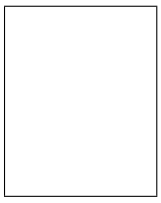
以上のような動作行動の開発は，シミュレーションに基づいた動作生成を必要とするコンピュータグラフィクス，ゲーム，バーチャルリアリティ等の分野でも必要であり研究開発が続けられている．これらを上手く組み合わせ，小さな努力で良い開発環境を得ることが重要になるだろう．我々は，オープンソースのモデリングツールである Blender とスクリプト言語である Python を利用することで，ブラックボックスを作らずに開発量を激減させる事ができた．

筆者らはゲーム，バーチャルリアリティ分野 [19] から研究を始めたが，ロボットの動作生成 [20] にも利用している．本格的なロボットの研究開発で利用が進めば，様々な要求や問題点が現れ得ると考えている．今後は，物理エンジンや動作生成の研究を進めると同時にそのような研究グループとの協力を摸索していきたい．

参 考 文 献

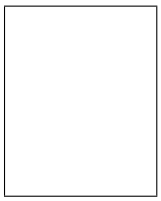
[1] J. K. Hahn: “Realistic Animation of Rigid Bodies” ACM SIGGRAPH Computer Graphics, Vol. 22 Issue 4, pp.299-308, 1988.
[2] B. Mirtich: “Impulse-Based Dynamics for Rigid-Body Simulation”, PhD thesis, University of California, 1996.
[3] J.J. Moreau: “Unilateral contact and dry friction in finite freedom dynamics”,International Centre for Mechanical Sciences Courses and Lectures vol. 302, Springer, Vienna, 1988.
[4] D. Baraff：“Analytical Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies”, Computer Graphics Proceedings(SIGGRAPH 89) Vol. 23, pp. 223-232, 1989.
[5] D. Baraff：“Fast Contact Force Computation for Non-penetrating Rigid Bodies”, Computer Graphics Proceedings(SIGGRAPH94), pp. 23-34, 1994.
[6] D. E. Stewart and J. C. Trinkle: “An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. Internat.” J. Numer. Methods Engineering, 1996.
[7] M. Moore and J. Wilhelms: “Collision Detection and Response for Computer Animation”, ACM SIGGRAPH Computer Graphics, Vol. 22 Issue 4, pp.289-298, 1988.
[8] K. Yamane, Y. Nakamura: “Efficient Parallel Dynamics Computation of Human Figures”, IEEE International Conference on Robotics and Automation, pp.530-537, 2002.
[9] R. Smith et.al: “Open Dynamics Engine”, http://www.ode.org/, 2000.
[10] E. Coumans et.al: “Bullet Physics”, http://bulletphysics.org, 2005.
[11] 長谷川晶一, 田崎勇一, 三武裕玄 他:“Springhead2”. http://springhead.info/, 2005
[12] E. G. Gilbert, D. W. Johnson, S. S. Keerthi: “A fast procedure for computing the distance between complex objects in three-dimensional space.”, IEEE Journal of Robotics and Automation, Vol.4, No.2, 1988.
[13] M. C. Lin and J. F. Canny: “A fast algorithm for incremental distance calculation”, Proc. of Intl. Conf. on Robotics and Automation 1991, Vol.2, pp 1008–1014, 1991.
[14] S. Hasegawa, M. Sato: “Real-time Rigid Body Simulation for Haptic Interactions Based on Contact Volume of Polygonal Objects”, Computer Graphics Forum, Vol. 23, Issue 3, pp.529–538, 2004.
[15] M. Anitescu, F. A. Potra: “A Time-Stepping Method for Stiff Multibody Dynamics with Contact and Friction”, International Journal for Numerical Methods in Engineering, Vol.55, pp.753-784, 2002.
[16] R. Featherstone: “Robot Dynamics: Equations and Algorithms”, IEEE International Conference on Robotics and Automation, pp. 826-834, 2000.
[17] T. Tokizaki, Y. Tazaki, H. Mitake, S. Hasegawa:“Pliant Motion: Integration of Virtual Trajectory Control into LCP Based Physics Engines”, Proc. SIGGRAPH 2009 Posters, #10, 2009.
[18] Blender Foundation, http://www.blender.org/documentation/

250PythonDoc
[19] H. Mitake, S. Hasegawa, M. Sato : “Reactive Virtual Creatures for Dexterous Physical Interactions”, 4th Intl. Conf. Motion in Games 2011 Proc., 2011.
[20] Y. Yamashita, T. Ishikawa, H. Mitake, I. Susa, F. Kato, Y. Takase, W. Seshimo, Y. Takehana, S. Onohara, T. Harano, S. Hasegawa, M. Sato : “Stuffed Toys Alive! Cuddly Robots From a Fantasy World”, SIGGRAPH 2012 Emerging Technologies, 2012.



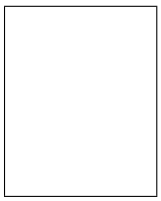
長谷川 晶一 (Shoichi Hasegawa)

1974 年 6 月 10 日生．1999 年東京工業大学大学大学院知能システム科学専攻修士終了．同年ソニー株式会社入社．2000 年東京工業大学精密工学研究所助手．2007 年電気通信大学知能機械科准教授．2010 年東京工業大学精密工学研究所准教授現在に至る．日本バーチャルリアリティ学会，日本ロボット学会，計測自動制御学会，情報処理学会各会員．バーチャルリアリティ，物理エンジン，力触覚，ヒューマンインタフェース，エンタテインメント工学の研究に従事．博士 (工学)．（日本ロボット学会正会員）



三武裕玄 (Hironori Mitake)

1984 年 1 月 12 日生．2011 年 東京工業大学大学院知能システム科学専攻博士修了．2008 年 4 月より 2011 年 3 月まで日本学術振興会特別研究員．2011 年より東京工業大学精密工学研究所助教，現在に至る．日本バーチャルリアリティ学会，日本ロボット学会，情報処理学会各会員．キャラクタ動作生成，バーチャルリアリティの研究に従事．博士（工学）．（日本ロボット学会正会員）



田崎 勇一 (Yuichi Tazaki)

1980 年 5 月 21 日生．2008 年 3 月東京工業大学大学院情報理工学研究科情報環境学専攻博士課程修了．2007 年 4 月より 2009 年 3 月まで日本学術振興会特別研究員．2008 年，Honda Research Institute Europe（ドイツ）客員研究員．2009 年より名古屋大学大学院工学研究科機械理工学専攻助教，今日に至る．二足歩行ロボットの制御，多自由度ロボットの動作計画などに従事．計測自動制御学会，IEEE 会員．博士 (工学)．（日本ロボット学会正会員）