分子動力学法シミュレーションの Processing による視覚化

北村宜久

平成 25 年 2 月 16 日

概 要

本研究では Processing による原子の動的挙動の視覚化を目的とする．第一原理計算ソフト VASP は原子系のエネルギーを精密に計算するが，計算時間がかかってしまう．そこで計算時間を短縮するため，分子動力学法（molecular dynamics，略して MD）を用い，Ruby でプログラムを開発した．そのプログラムから出力される原子座標を Processing で読み取り視覚化した．また，今後実用することを考えて VASP とほぼ同じ仕様の pseudo VASP の OUTCAR から原子座標を読み取り視覚化した．Maple と Maya の視覚化ソフトと比較した．

目 次

第1章　背景

西谷研究室では第一原理計算ソフト VASP を用いて，材料の物性を予測する様々なシミュレーションを行い，視覚化している．視覚化することで物体の動きが見え，理解しやすい．

シュミレーション結果を視覚化するために結晶モデル構築ソフト Maya や数値計算ソフト Maple を利用してきた．Maya は 3D アニメーションに関連する機能は充実しているが，ライセンス料がとても高価である．また Maple は数値計算ソフトとしては問題なく，グラフ描画機能においては優れているが，アニメーション機能が乏しいことや処理時間がかかるという欠点がある．そこで本研究ではそれらの欠点がない Processing を使用する．Processing はグラフィック機能を中心とした Java を単純化した言語で，画像処理やアニメーションに特化している．さらにオープンソースであり，処理速度も速い．本研究では Processing を用いてシミュレーションした原子座標を読み取り，視覚化することを目的とする．

VASP は原子系のエネルギーを精密に計算するが，計算時間がかかってしまう．そこで今回のシミュレーションにはコントローラ開発を目的に開発された pseudo VASP や自らが開発した分子動力学法を用いたプログラムを用い視覚化する．その結果を Maple と Maya で比較する．

第2章　方法

2.1　使用言語

今回の研究において 2 つのプログラミング言語を使用した．それぞれの長所を活用することで，プログラムの簡易化を可能にした．以下にその使い分けを示す．

2.1.1　Ruby

Ruby はまつもとゆきひろ氏のによって開発されたオブジェクト指向のスクリプト言語の名称である．全てのデータは一貫してオブジェクトとして表現されているので考えたことを直感的に記述できる．また継承や Mix-in といった，オブジェクト指向らしい機能も備わっている．また，様々なクラスライブラリが標準で添付さてれいるほか，例外によるエラー処理や自動的にメモリの開放を行うガベージコレクタなど，快適なプログラミングを支援する機能も備わっている．今回の研究ではこの Ruby 言語を用いて分子動力学プログラムを開発している．

2.1.2　Processing

Processing は，アートとデザインのためのプログラミングを実現，オープンソース開発環境として Ben Fly 氏 Casey Reas 氏によって開発されました．Windows にも Mac にもさらに Linux にも対応していて，無料でダウンロードして利用することができる．シンプルで軽く，はじめての人でも使いやすい環境である．今回の研究では視覚化のために Processing を用いる．

2.2　分子動力学

分子動力学法は系を構成する粒子の運動方程式を時間について離散化し，それらの方程式を連立して解いて粒子の運動を追跡していく方法である．ニュートンの運動方程式はエネルギー保存則を満たす．したがって，熱力学的平衡状態を対象としたシミュレーションの場合には小正準集団に対してのみ適用することができる．

2.2.1 基礎方程式

系のエネルギーが保存される小正準集団や非平衡状態に対するシミュレーションに際しては，ニュートンの運動方程式が用いられる．粒子 i の位置ベクトルを r_i，粒子 i に作用する力を f_i とすると，ニュートンの運動方程式は次のように書ける．

$$m\frac{d^2r_i}{dt^2} = f_i \tag{2.1}$$

速度ベクトル v_i は位置の微分から，

$$v_i = \frac{dr_i}{dt} \tag{2.2}$$

もし，外力がなければ，系の運動エネルギーや運動量が保存される．しかしながら，シミュレーションでは一般的に有限のシミュレーション領域を設定し，境界の影響を少なくするために周期的境界条件を用いるので，必ずしもこれらの量が保存されるとは限らない．

2.2.2 Verlet のアルゴリズム

Verlet のアルゴリズムは初期状態以外では全く速度を用いないで粒子を移動させる特徴がある．Verlet 法は式 (2.1) から直接粒子の位置の時間発展を求める差分方程式をつくる．時刻 $t+h$ と時刻 $t-h$ における粒子の位置 $r_i(t \pm h)$ をテーラー級数展開し，式 (2.2) と

$$\dot{r}_i = \frac{dr_i}{dt} \tag{2.3}$$

を用いると，

$$r_i(t+h) = r_i(t) + h\dot{r}_i(t) + h^2\frac{f_i(t)}{2m} + O(h^3) \tag{2.4}$$

$$r_i(t-h) = r_i(t) - h\dot{r}_i(t) + h^2\frac{f_i(t)}{2m} + O(h^3) \tag{2.5}$$

を得る．両式の和と差をとると，

$$r_i(t+h) + r_i(t-h) = 2r_i(t) + h^2\frac{f_i(t)}{2m} + O(h^4) \tag{2.6}$$

$$r_i(t+h) - r_i(t-h) = 2h\dot{r}_i(t) + O(h^2) \tag{2.7}$$

これより時刻 $t+h$ における位置と t における速度は

$$r_i(t+h) = 2r_i(t) - r_i(t-h) + h^2\frac{f_i(t)}{m} + O(h^4) \tag{2.8}$$

$$\dot{r}_i(t) = \frac{1}{2h}[r_i(t+h) - r_i(t-h)] + O(h^2) \tag{2.9}$$

が与えられる．これが Verlet の差分式である．時刻 $t+h$ における位置を求めるには 2 つの時刻 t と $t-h$ での位置が必要である．初期条件と位置を速度で与えると，$t=h$ における位置 $r_i(h)$ は式 (2.8) から求まる．これと $r_i(0)$ から $r_i(2h)$ を計算し，式 (2.9) より速度 $\dot{r}_i(h)$ が得られる．これを逐次的に解くことにより粒子の運動がシミュレーションできる．

2.2.3 分子間ポテンシャル

シミュレーションを行うためにはモデルを定めなければならない．分子は球形で化学的に不活性であり，分子間に働く力はそれらの距離のみに依存すると仮定する．この場合，全ポテンシャルエネルギー U は 2 粒子相互関係の和

$$U = u(r_{12}) + u(r_{13}) + \cdots + u(r_{23}) + \cdots = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u(r_{ij}) \tag{2.10}$$

となり，ここで $u(r_{ij})$ は粒子 i, j 間の距離 r_{ij} のみ依存する．電気的に中性な分子についての $u(r)$ の形は，原理的には量子力学な計算により第一原理から構成できるものの，その計算は非常に困難である．単純液体についての $u(r)$ の最も重要な特徴は r が小さいところでの強い斥力と r が大きいところでの弱い引力である．小さな r での斥力はパウリの排他的原理によるものである，つまり，2 つの分子の電子雲は重なりをさけるために変形しなけれえばならなず，結果として電子のいくつかは異なる量子状態に入ることとなる．これらの効果により運動エネルギーが増大して，電子間にはコア斥力と呼ばれる実効的な斥力が生じる．r が弱いときに支配的となる弱い引力は，各分子が互いに分極を起こすことによるものであり，この引力はファン・デル・ワールス力と呼ばれている．最もよく使われている $u(r)$ の形の 1 つはレナードジョーンズポテンシャルである．図 2.2.3 にレナードジョーンズ・ポテンシャルを示す.

相互作用の斥力部分の $r^{(-12)}$ という形は便宜上選んだものである．レナードジョーンズ・ポテンシャルは長さ σ とエネルギー ϵ という 2 つのパラメータを持っている．$r=\sigma$ で $u(r)=0$ となり，$r>3\sigma$ はほとんど 0 になる．パラメータ ϵ は $u(r)$ の極小点でのポテンシャルの深さである，極小は $r=2^{\frac{1}{\sigma}}\sigma$ のところに生じる．ポテンシャルに生じる力は，ポテンシャルを微分することで得られる．式 (2.10) より分子 i に加わる F_i は，

$$F_i = -\frac{\partial U}{\partial r_i} = -\sum_{j}^{N} \frac{\partial u(r_{ij})}{\partial r_i} \tag{2.11}$$

という偏微分で表される．U を r_i で偏微分するにあたり，このままでは計算が困難なので，以下の関係式を導入する．

$$\frac{\partial r_{ij}}{\partial r_i} = -\frac{\partial r_{ij}}{\partial r_{ij}} \tag{2.12}$$

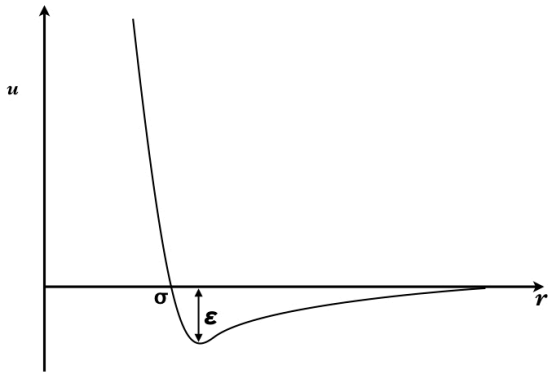

図 2.1: レナードジョーンズポテンシャル.

この式を用いると，F_i は

$$F_i = -\sum_j^N \frac{\partial u(r_{ij})}{\partial r_{ij}} \frac{\partial r_{ij}}{\partial r_i} = \sum_j^N \frac{\partial u(r_{ij})}{\partial r_{ij}} \frac{r_{ij}}{r_{ij}} \tag{2.13}$$

となり，r_{ij} に対する偏微分は

$$\frac{du(r)}{dr} = -24\frac{\epsilon}{\sigma}\left\{2(\frac{\sigma}{r})^{13} - (\frac{\sigma}{r})^7\right\} \tag{2.14}$$

となる.

2.3 Ruby による MD の実装

Ruby によって MD を実装した．以下にコードの説明を示す.

2.3.1 初期値の設定

粒子の初期配置を指定する．全粒子に 1〜12 の番号を付け x 軸，y 軸に数値を入力する．運動する粒子は 1〜4 の番号で 5〜12 の番号は固定する．これにより全粒子が配置される．speed() は運動する粒子 4 つのみ設定し，速度計算に用いられる．(speed() に関しては付録参照)

```
$init = [1,[1,1],[speed(),speed()],[2,3,5,12]],
        [2,[2,1],[speed(),speed()],[1,4,6,7]],

        [11,[0,2]],
```

```
 [12,[0,1]]

for i in 0..11 do
 $x[i] = $init[i][1][0]
 $y[i] = $init[i][1][1]
 $tmp << [$x[i], $y[i]]
end
```

同じく初期速度を設定する．こちらは運動する粒子のみ設定する．vx,vy には speed() からランダムな数値が入り初期速度となる．ax,ay は加速度を表しているが，初期状態では常に 0 を指定しておく．

```
for i in 0..3 do
 $vx[i] = $init[i][2][0]
 $vy[i] = $init[i][2][1]
 $ax[i] = 0
 $ay[i] = 0
end
```

以上が初期配置を初期速度の設定となる．

2.3.2 レナード-ジョーンズ・ポテンシャル計算

先ほどの分子間ポテンシャルの式から計算を行い，ポテンシャルに生じる力を算出する．

```
r2 = $dx*$dx + $dy*$dy
rm2 = 1/r2
rm6 = rm2*rm2*rm2
kk = 24*rm6*(2*rm6-1)*rm2
fxij = kk*$dx
fyij = kk*$dy
$dx = fxij
$dy = fyij
```

これらを force1 として Verlet のアルゴリズムで使用する．

2.3.3 Verlet のアルゴリズム

Verlet のアルゴリズムレナードジョーンズポテンシャルにより算出された粒子間に働く力をもとに，粒子を逐次的に移動させる．force1 から呼び出した dx,dy

を代入し，新しく ax,ay を求める．これらの計算はニュートンの第 3 法則によるものである．

```
force1()
$ax[i] = $ax[i] + $dx
$ay[i] = $ay[i] + $dy
```

以上を force2 として，以下で使用する．

```
for i in 0..3 do
  $x[i] = $x[i] + $vx[i]*$dt + 0.5*$ax[i]*$dt2
  $y[i] = $y[i] + $vy[i]*$dt + 0.5*$ay[i]*$dt2
  $vx[i] = $vx[i] + 0.5*$ax[i]*$dt
  $vy[i] = $vy[i] + 0.5*$ay[i]*$dt
end
force2()
for i in 0..3 do
  $vx[i] = $vx[i] + 0.5*$ax[i]*$dt
  $vy[i] = $vy[i] + 0.5*$ay[i]*$dt
end
```

以上を verlet とし，粒子の位置を算出する．ここで算出された x,y が粒子の座標になり，時間刻みで表示される．さらに加速度から速度の更新もしている．これを時間刻みで行うことで，粒子の位置を時間経過とともに算出していく．

これらの座標を配列に格納し，Processing で読み込むために tsv というファイルを出力する．

2.4 MD の Processing による視覚化

Ruby のプログラムで作成した原子座標が書かれているファイルを次の Processing でつくったプログラムに読み込ませることでモデルとして表示させる．Processing で作ったコードには立体的に原子の奥行きがわかる 3 次元と全ての原子を平面上に示した 2 次元の 2 種類作った．さらに動いている原子の速度によって速ければ赤，遅ければ青になるようなグラデーションを実装した．

2.4.1 データの読み込み

データを読み込む為に FloatTable というクラスを作成した.（詳しくは付録を参照）データを読み込み data_point の行列にそれぞれ格納する．データの値が小さいので視覚化した時に見やすいように size 倍している．今回は 400 倍している．

```
    data = new FloatTable("ファイル名");
    for (int row = 0; row < data_row-1; row++) {
for (int col = 0; col < data_col; col++) {
    data_point[row][col] = size*data.getFloat(row, col);
}
    }
```

2.4.2 速度の計算

MD のプログラムで時間おきに座標を記録しているので各座標の差を取り，移動距離を算出する．つまり速度を求めることに等しい．また，ループしても違和感のないように最初と最後も差を取り移動距離を算出し，z に格納した．

```
for (int i=0;i<28;i++){
  for(int j=0;j<12;j++){
    xa = data_point[i][j*2] - data_point[i+1][j*2];
    ya = data_point[i][j*2+1] - data_point[i+1][j*2+1];
    za = xa*xa + ya*ya;
    z[i][j] = sqrt(za);
  }
}
for(int i=0;i<12;i++){
  xa = data_point[29][i*2] - data_point[0][i*2];
  ya = data_point[29][i*2+1] - data_point[0][i*2+1];
  za = xa*xa + ya*ya;
  z[29][i] = sqrt(za);
}
```

2.4.3 速度からグラデーションを作り，視覚化する

先ほどの配列 z から最大値と最小値を算出し，それを l に入れる．for 文により全ての z に対して最小値を引き，l で割った値の色の最大値 255 を掛けた値を R に入れる．青にするためには，`1-(nz-z_min)/l)` というように赤で示した時の分子を 1 から引くことにより示すことができる．fill(R,0,B) とすることで原子が速くなればなるほど赤になるというようになっている．(図 2.4.3) `count = (count+1)%(data_row-1)` をすることで無限ループさせ，表示させている．

```
for (int i=0;i<29;i++) {
  for(int j=0;j<12;j++){
```

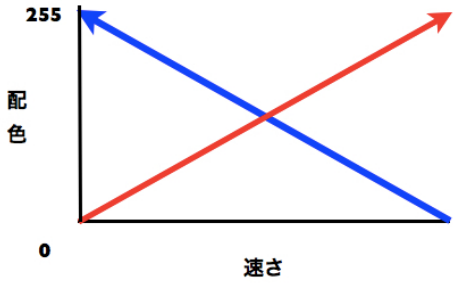

図 2.2: 色の変化を図的に表したもの.

```
    if (z_max<z[i][j]){
z_max=z[i][j];}
    if (z_min>z[i][j]){
z_min=z[i][j];}
    }
 }
  l = z_max-z_min;
  for (int i=0;i<12;i++){
    nz = z[count][i];
    R = int(255*(nz-z_min)/l);
    B = int(255*(1-(nz-z_min)/l));
    fill(R,0,B);
    ellipse(data_point[count][i*2]/2+10,data_point[count][i*2+1]/2+10,20,20)
  }
  count = (count+1)%(data_row-1);
}
```

2.5 pseudo VASP の Processing による視覚化

pseudo VASP の出力ファイルである OUTCAR には 3 次元の座標が書かれているので，先ほどのコードを 3 次元に拡張する必要がある．主な変更点は 2 次元配列を 3 次元配列に書き換えるだけである.（詳しくは付録を参照）

2.5.1 pseudoVASP

中井が開発した pseudo VASP の出力ファイルを視覚化する．pseudo VASP は VASP を用いて第一原理計算を行うよりも短い時間で，原子のもつネルギーやフォースを求めるのでコントローラの開発に適している． pseudo VASP では，Lennard-Jones ポテンシャルを用いて系のエネルギーを求め，求めたエネルギーを微分することにより各原子のもつフォースを計算していく．なお，pseudo VASP の入出力ファイルは VASP で使用するものと同じである．

2.5.2 視点の切り替え

3 次元に拡張することで正面からの視点だけでは全体の原子の動きが見づらくなってしまう．そこで視点を切り替えることでよりシミュレーションの見やすさを追求した．そこでモデルの回転，拡大するコードを加えた．以下の if 文のコードによりマウスを押しながら動かすことでモデルの向きが変わる．回転させる方法としては rotate() を使うことで () 内の回転角度分回すことができる．そして，キーボードの p を押すことで depth の値が 10 ずつ増え，より奥に視点が近づくようになっている．反対に n を押せば depth の値が 10 ずつ減り視点が遠ざかるようになっている．

```
camera(0, 0, depth/20, 0, 0, 0, 0, 1, 0);
if (mousePressed) {
  mvX=mouseX-oldX;
  mvY=mouseY-oldY;
  oldX=mouseX;
  oldY=mouseY;
}
else {
  mvX=0;
  mvY=0;
  oldX=mouseX;
  oldY=mouseY;
}
nowX += mvX;
nowY -= mvY;
if(keyPressed) {
  if(key=='p'){
  depth+=10;
  }
  else if (key=='n'){
```

```
    depth-=10;
  }
}
rotateX(radians(nowY/3));
rotateY(radians(nowX/3));
```

第3章　結果と考察

3.1　MD の視覚化

図 3.1 は Maple により視覚化された結果，図 3.1 は Processing により視覚化された結果である．これらを比較すると，2 つの相違点がわかる．まず，Processing 方が見た目がよく，色や質感の表現に優れている．また速度変化の視覚化が大きな違いである．青色は速度 0 の状態を示し，速度が上がるにつれて赤色へと変化している．アニメーションで再生すると，色の変化がグラデーションでスムーズに変化していることが分かり，レンダリングのクオリティが向上したことが分かる．

3.2　pseudo VASP の視覚化

図 3.2 は Maya により視覚化された結果，図 3.2 は Processing により視覚化された結果である．2 つの図は共に正面から見たものである．視点の切り替え機能は共に備わっている．これらを比較するとあまり大きな違いはないが，アニメーションで再生すると大きな違いがある．Maya の方がスムーズに動いていることがわかる．以上のことから Maya の方がキレイであることがわかる．

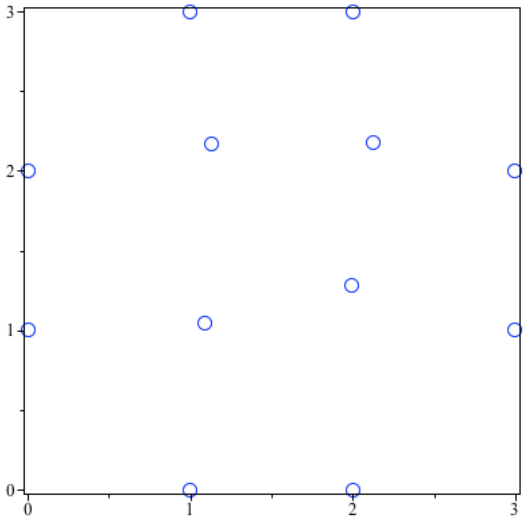

図 3.1: Malpe により視覚化した結果.

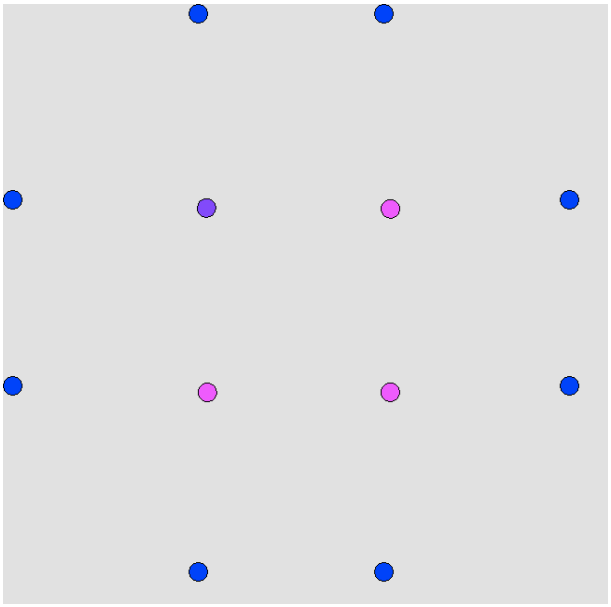

図 3.2: Processing により視覚化した結果.

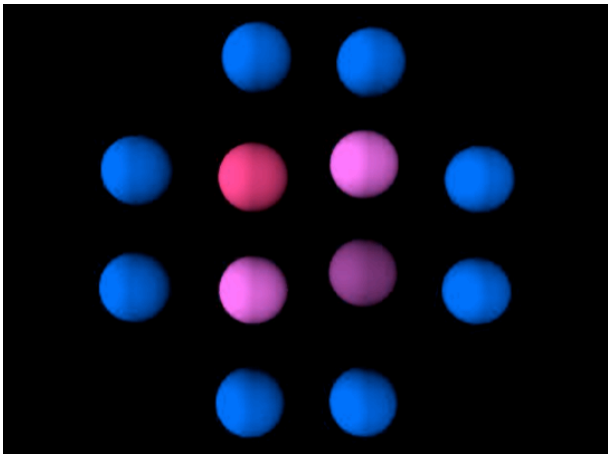

図 3.3: Maya により視覚化した結果.

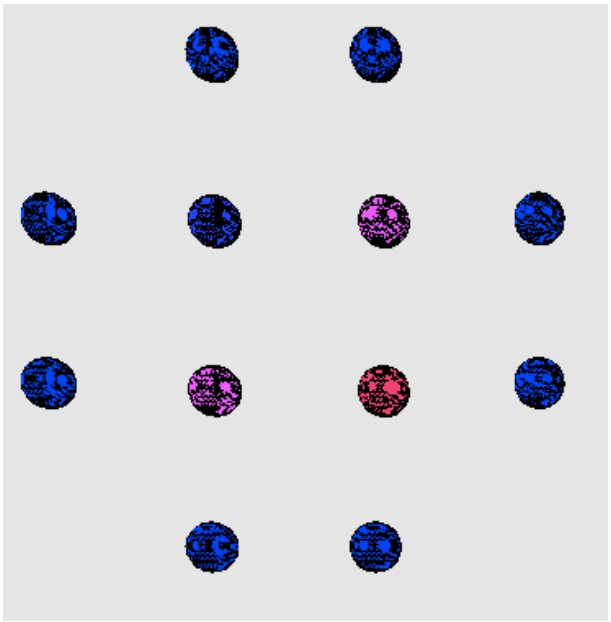

図 3.4: Processing により視覚化した結果.

第4章　総括

西谷研究室では第一原理計算ソフト VASP を用いてシュミレーションを行い，視覚化している．視覚化することで物体の動きが見え，理解しやすい．シミュレーションを視覚化するために Processing のプログラムを作成した．また，シュミレーションを行うソフトとして，分子動力学法を用いて作成した Ruby プログラムと pseudo VASP を使用した．

Maple と比較するとの Processing の特長であるアニメーション機能を生かしていることがわかる．アニメーションで再生すると，原子の速さによりグラデーションでスムーズに変化していることがわかる．

Maya と比較すると，視点の切り替え機能やアニメーションによる原子の動きなど Processing でほぼ再現できている．しかし，Maya の方がよりスムーズに動きキレイである．

これらの比較を図 **??**にまとめた．Maple は数値計算ソフトとしては優秀だが，視覚化することを目的とするならば扱いづらいことがわかる．Maya は動きやキレイさであるアニメーション機能は充実している．しかし，読み込むプログラムの量が多いと途中で勝手に改行され，正常に読み込まれない場合があるなどコード開発が非常に困難である．Processing はコード開発が最も容易である．動き，キレイさなどのアニメーションはある程度であれば実装できることがわかる．

	コード開発	動き	キレイさ
Maple	×	×	△
Maya	×	◎	◎
Processing	◎	○	○

図 4.1: それぞれの性能比較.

第5章　謝辞

本研究を遂行するにあたり，終始多大なる有益なご指導，及び丁寧な助言を頂いた西谷滋人教授に深い感謝の意を表します．

また，本研究を進めるにつれ，西谷研究室員の皆様にも様々な知識の供給，ご協力頂き，本研究を大成する事ができました．最後になりましたが，この場を借りて心から深く御礼申し上げます．

第6章　付録

Ruby MD

```
include Math

def speed()
  10*rand()/(10**12)-1
end

$init = [1,[1,1],[speed(),speed()],[2,3,5,12]],
        [2,[2,1],[speed(),speed()],[1,4,6,7]],
        [3,[1,2],[speed(),speed()],[1,4,10,11]],
        [4,[2,2],[speed(),speed()],[2,3,8,9]],
        [5,[1,0]],
        [6,[2,0]],
        [7,[3,1]],
        [8,[3,2]],
        [9,[2,3]],
        [10,[1,3]],
        [11,[0,2]],
        [12,[0,1]]
$x = []
$y = []
$ax = []
$ay = []
$vx = []
$vy = []
$dt = 0.01
$dt2 = $dt*$dt
$tmp = []

def force1()
  r2 = $dx*$dx + $dy*$dy
```

```
  rm2 = 1/r2
  rm6 = rm2*rm2*rm2
  kk = 24*rm6*(2*rm6-1)*rm2
  fxij = kk*$dx
  fyij = kk*$dy
  $dx = fxij
  $dy = fyij
end

def force2()
  for i in 0..3 do
    $ax[i] = 0
    $ay[i] = 0
  end
  for i in 0..3 do
    for j in 0..3 do
      jj = $init[i][3][j]-1
      $dx = $x[i] - $x[jj]
      $dy = $y[i] - $y[jj]
      force1()
      $ax[i] = $ax[i] + $dx
      $ay[i] = $ay[i] + $dy
    end
  end
end

def verlet()
  for i in 0..3 do
    $x[i] = $x[i] + $vx[i]*$dt + 0.5*$ax[i]*$dt2
    $y[i] = $y[i] + $vy[i]*$dt + 0.5*$ay[i]*$dt2
    $vx[i] = $vx[i] + 0.5*$ax[i]*$dt
    $vy[i] = $vy[i] + 0.5*$ay[i]*$dt
  end
  force2()
  for i in 0..3 do
    $vx[i] = $vx[i] + 0.5*$ax[i]*$dt
    $vy[i] = $vy[i] + 0.5*$ay[i]*$dt
  end
end
```

```
def show()
  for i in 0..11 do
     $tmp << [$x[i],$y[i]]
  end
end

for i in 0..11 do
  $x[i] = $init[i][1][0]
  $y[i] = $init[i][1][1]
  $tmp << [$x[i], $y[i]]
end

\begin{enumerate}
\item 初期速度の設定
\end{enumerate}
for i in 0..3 do
  $vx[i] = $init[i][2][0]
  $vy[i] = $init[i][2][1]
  $ax[i] = 0
  $ay[i] = 0
end

for i in 0..29 do
  for j in 0..3 do
    verlet()
  end
  $tmp << show()
end
$tmp.delete(0..11)

def dummy(file,n)
  a = 0.to_s
  for i in 0..n-1 do
    a += "\t" + 0.to_s
  end
  file.print a+"\n"
end
```

```
data = []
l = $tmp.length
file = open("sample.tsv","w")
dummy(file,24)
for i in 0..l-1 do
  if i%12==0 && i!=0 then
    file.print "\n"
    file.print 0.to_s+"\t"
  end
  if i==0 then
    file.print 0.to_s+"\t"
  end
  for j in 0..1 do
    file.print $tmp[i][j].to_s+"\t"
  end
end
```

Processing FloatTable

```
class FloatTable {
  int rowCount;
  int columnCount;
  float[][] data;
  String[] rowNames;
  String[] columnNames;


  FloatTable(String filename) {
    String[] rows = loadStrings(filename);

    String[] columns = split(rows[0], TAB);
    columnNames = subset(columns, 1); // upper-left corner ignored
    scrubQuotes(columnNames);
    columnCount = columnNames.length;

    rowNames = new String[rows.length-1];
    data = new float[rows.length-1][];

    // start reading at row 1, because the first row was only the column heade
    for (int i = 1; i < rows.length; i++) {
```

```
    if (trim(rows[i]).length() == 0) {
      continue; // skip empty rows
    }
    if (rows[i].startsWith("#")) {
      continue;  // skip comment lines
    }

    // split the row on the tabs
    String[] pieces = split(rows[i], TAB);
    scrubQuotes(pieces);

    // copy row title
    rowNames[rowCount] = pieces[0];
    // copy data into the table starting at pieces[1]
    data[rowCount] = parseFloat(subset(pieces, 1));

    // increment the number of valid rows found so far
    rowCount++;
  }
  // resize the 'data' array as necessary
  data = (float[][]) subset(data, 0, rowCount);
}


void scrubQuotes(String[] array) {
  for (int i = 0; i < array.length; i++) {
    if (array[i].length() > 2) {
      // remove quotes at start and end, if present
      if (array[i].startsWith("\"") && array[i].endsWith("\"")) {
        array[i] = array[i].substring(1, array[i].length() - 1);
      }
    }
    // make double quotes into single quotes
    array[i] = array[i].replaceAll("\"\"", "\"");
  }
}


int getRowCount() {
```

```
  return rowCount;
}


String getRowName(int rowIndex) {
  return rowNames[rowIndex];
}


String[] getRowNames() {
  return rowNames;
}


// Find a row by its name, returns -1 if no row found.
// This will return the index of the first row with this name.
// A more efficient version of this function would put row names
// into a Hashtable (or HashMap) that would map to an integer for the row.
int getRowIndex(String name) {
  for (int i = 0; i < rowCount; i++) {
    if (rowNames[i].equals(name)) {
      return i;
    }
  }
  //println("No row named '" + name + "' was found");
  return -1;
}


// technically, this only returns the number of columns
// in the very first row (which will be most accurate)
int getColumnCount() {
  return columnCount;
}


String getColumnName(int colIndex) {
  return columnNames[colIndex];
}
```

```
String[] getColumnNames() {
  return columnNames;
}


float getFloat(int rowIndex, int col) {
  // Remove the 'training wheels' section for greater efficiency
  // It's included here to provide more useful error messages

  // begin training wheels
  if ((rowIndex < 0) || (rowIndex >= data.length)) {
    throw new RuntimeException("There is no row " + rowIndex);
  }
  if ((col < 0) || (col >= data[rowIndex].length)) {
    throw new RuntimeException("Row " + rowIndex + " does not have a column
  }
  // end training wheels

  return data[rowIndex][col];
}


boolean isValid(int row, int col) {
  if (row < 0) return false;
  if (row >= rowCount) return false;
  //if (col >= columnCount) return false;
  if (col >= data[row].length) return false;
  if (col < 0) return false;
  return !Float.isNaN(data[row][col]);
}


float getColumnMin(int col) {
  float m = Float.MAX_VALUE;
  for (int i = 0; i < rowCount; i++) {
    if (!Float.isNaN(data[i][col])) {
      if (data[i][col] < m) {
```

```
        m = data[i][col];
      }
    }
  }
  return m;
}


float getColumnMax(int col) {
  float m = -Float.MAX_VALUE;
  for (int i = 0; i < rowCount; i++) {
    if (isValid(i, col)) {
      if (data[i][col] > m) {
        m = data[i][col];
      }
    }
  }
  return m;
}


float getRowMin(int row) {
  float m = Float.MAX_VALUE;
  for (int i = 0; i < columnCount; i++) {
    if (isValid(row, i)) {
      if (data[row][i] < m) {
        m = data[row][i];
      }
    }
  }
  return m;
}


float getRowMax(int row) {
  float m = -Float.MAX_VALUE;
  for (int i = 1; i < columnCount; i++) {
    if (!Float.isNaN(data[row][i])) {
      if (data[row][i] > m) {
```

```
          m = data[row][i];
        }
      }
    }
    return m;
  }


  float getTableMin() {
    float m = Float.MAX_VALUE;
    for (int i = 0; i < rowCount; i++) {
      for (int j = 0; j < columnCount; j++) {
        if (isValid(i, j)) {
          if (data[i][j] < m) {
            m = data[i][j];
          }
        }
      }
    }
    return m;
  }


  float getTableMax() {
    float m = -Float.MAX_VALUE;
    for (int i = 0; i < rowCount; i++) {
      for (int j = 0; j < columnCount; j++) {
        if (isValid(i, j)) {
          if (data[i][j] > m) {
            m = data[i][j];
          }
        }
      }
    }
    return m;
  }
}
```

Processing MD2D

```
FloatTable data;
float[][] data_point = new float[30][24];//tsv_gyou need improve number[row][c
int size=400;
int count=0;
int maxFrame=20;
int data_row=30; //contain first 0_row
int data_col=24; //not contain first 0_col
int R, B;
float xa=0, ya=0, za=0;
float l, z_max, z_min, nz;
float[][] z = new float[30][12];

void setup() {
    size (800, 800);
    colorMode(RGB, 100);
    background(90);
    translate(width/2,height/2,0);
    translate(0,size*zoom,size*zoom);
    frameRate(10);
    data = new FloatTable("sample.tsv");
    for (int row = 0; row < data_row-1; row++) {
for (int col = 0; col < data_col; col++) {
    data_point[row][col] = size*data.getFloat(row, col);
}
    }
    for (int i=0;i<28;i++){
      for(int j=0;j<12;j++){
        xa = data_point[i][j*2] - data_point[i+1][j*2];
        ya = data_point[i][j*2+1] - data_point[i+1][j*2+1];
        za = xa*xa + ya*ya;
        z[i][j] = sqrt(za);
      }
    }
    for(int i=0;i<12;i++){
      xa = data_point[29][i*2] - data_point[0][i*2];
      ya = data_point[29][i*2+1] - data_point[0][i*2+1];
      za = xa*xa + ya*ya;
      z[29][i] = sqrt(za);
    }
```

```
    z_max = z[0][0];
    z_min = z[0][0];
    for (int i=0;i<29;i++) {
      for(int j=0;j<12;j++){
      if (z_max<z[i][j]){
  z_max=z[i][j];}
      if (z_min>z[i][j]){
  z_min=z[i][j];}
      }
   }
    l = z_max-z_min;
}

void draw() {
    background(90);
    for (int i=0;i<12;i++){
      nz = z[count][i];
      R = int(255*(nz-z_min)/l);
      B = int(255*(1-(nz-z_min)/l));
      fill(R,0,B);
      ellipse(data_point[count][i*2]/2+10,data_point[count][i*2+1]/2+10,20,20)
  }
  count = (count+1)%(data_row-1);
}
```

Processing VASP3D

```
int mvX=0, mvY=0, oldX=0, oldY=0, nowX=0, nowY=0;
int frame;
FloatTable data;
float[][] data_point = new float[10][96];//tsv_gyou need improve number[row][c
float depth=20000.0;
int size=400;
int count=0;
int maxFrame=20;
int data_row=10; //contain first 0_row
int data_col=95; //not contain first 0_col
int R, B;
float zoom=1.0,x=0.0;
float xa=0, ya=0, za=0, dis=0;
```

```
float l, dis_max, dis_min, ndis;
float[][] disa = new float[10][32];

void setup() {
    size (800, 800,P3D);
    colorMode(RGB, 100);
    background(90);
    translate(width/2,height/2,0);
    translate(0,size*zoom,size*zoom);
    frameRate(10);
    data = new FloatTable("sample.tsv");
    for (int row = 0; row < data_row-1; row++) {
for (int col = 0; col < data_col; col++) {
    data_point[row][col] = size*data.getFloat(row, col);
}
    }
    for (int i=0;i<9;i++){
      for(int j=0;j<32;j++){
        xa = data_point[i][j*3] - data_point[i+1][j*3];
        ya = data_point[i][j*3+1] - data_point[i+1][j*3+1];
        za = data_point[i][j*3+2] - data_point[i+1][j*3+2];
        dis = xa*xa + ya*ya + za*za;
        disa[i][j] = sqrt(dis);
      }
    }
    for(int i=0;i<32;i++){
      xa = data_point[9][i*3] - data_point[0][i*3];
      ya = data_point[9][i*3+1] - data_point[0][i*3+1];
      za = data_point[9][i*3+2] - data_point[0][i*3+2];
      dis = xa*xa + ya*ya + za*za;
      disa[9][i] = sqrt(dis);
    }
    dis_max = disa[0][0];
    dis_min = disa[0][0];
    for (int i=0;i<10;i++) {
      for(int j=0;j<32;j++){
      if (dis_max<disa[i][j]){
  dis_max=disa[i][j];}
      if (dis_min>disa[i][j]){
```

```
  dis_min=disa[i][j];}
      }
   }
    l = dis_max-dis_min;
}
void draw() {
  background(90);
  camera(0, 0, depth/20, 0, 0, 0, 0, 1, 0);
  if (mousePressed) {
    mvX=mouseX-oldX;
    mvY=mouseY-oldY;
    oldX=mouseX;
    oldY=mouseY;
   }
  else {
    mvX=0;
    mvY=0;
    oldX=mouseX;
    oldY=mouseY;
  }
  nowX += mvX;
  nowY -= mvY;
  if(keyPressed) {
    if(key=='p'){
    depth+=10;
    }
    else if (key=='n'){
      depth-=10;
    }
  }
  rotateX(radians(nowY/3));
  rotateY(radians(nowX/3));

    for (int i=0;i<32;i++){
      ndis = disa[count][i];
      R = int(255*(ndis-dis_min)/l);
      B = int(255*(1-(ndis-dis_min)/l));
      fill(R,0,B);
      pushMatrix();
```

```
        translate(data_point[count][i*3],data_point[count][i*3+1],data_point[cou
        sphere(15);
        popMatrix();
    }
    count = (count+1)%(data_row-1);
}
```

関連図書

[1] 西谷滋人監修, 「VASP マニュアル」 , http://ist.ksc.kwansei.ac.jp/ nishitani/?RecentPublications.

[2] 神山新一,「分子動力学シミュレーション」,（朝倉書店 1997）.

[3] 谷沖由香,「運動する粒子の温度の視覚化」,（関西学院大学 物理学部 物理学専攻 卒業論文 2007）.

[4] Ben Fry,「ビジュアライジング・データ ー Processing による情報視覚化手法」,（オライリー・ジャパン 2008）.

[5] Casey Reas (著), Ben Fry (著), 船田 巧 (翻訳) ,「Processing をはじめよう (Make: PROJECTS)」, (オライリージャパン 2011/10/22).