運動する粒子の温度の視覚化

関西学院大学　理工学部　物理学科　物理学専攻
金子研究室　3013 谷沖　由香

平成 20 年 7 月 31 日

概要

数値計算ソフト Maple ではシミュレーション表現に限界がある.CG アニメーションソフト Maya を使用することによりシミュレーションのクオリティを高め, 表現の幅を広げることが可能である. 本研究では分子動力学法を用いて, 運動する粒子の温度変化の視覚化を目的とし, 温度による色変化や 3 次元表現, 視点の切り替えによるシミュレーションの見やすさを追求した.

本研究では, 分子動力学法を用いて粒子の運動軌道を算出し,Maya による視覚化を行った. 今回は球状粒子を仮定し, 粒子の回転運動は考慮する必要はないものとした. 非平衡状態を対象としたシミュレーションを行うため, 系のエネルギーが保存される定エネルギー分子動力学による計算を行った. シミュレーションを計算アルゴリズムには, 初期配置および初期速度を与え, 一定の時間きざみで全粒子を移動させるステップを繰り返す方法,Verlet のアルゴリズムを用いたプログラムを作成した. その際,Ruby と MEL,2 つのスクリプト言語を使用した.Ruby は変数などの宣言が必要なく, 粒子の軌道計算や速度の算出など, 主に物理的な数値計算に使用した. 一方,MEL は独自に開発された Maya 専用の言語であり, アニメーションの設定に関するプログラムに使用した.2 つの言語を使い分けることでそれぞれの長所を活用し, プログラムの簡易化を可能にした.

目 次

第1章　序論

1.1　研究の背景と目的

西谷研究室では結晶モデル構築ソフト Medea や数値計算ソフト Maple を利用してきたが, 視覚表現に乏しく, アニメーションとしてのクオリティは低い. そこで,CG アニメーションソフト Maya を使用することによりアニメーションのクオリティを高め, 視覚表現の幅を広げることを可能にした.

分子動力学法 (molecular dynamics method, 略して MD) は, モンテカルロ法 (Monte Carlo method, 略して MC) と並ぶ分子シミュレーション (molecular simulation) 法であり, 粒子の運動方程式に従って運動を追跡していく決定論的方法である. 熱力学的平衡状態に対するシミュレーション法であるモンテカルロ法に対し, 分子動力学法は熱平衡状態はもちろんのこと, 工学的に重要な非平衡な現象にも適用できるので, 非常に適用範囲が広いシミュレーション法と言える.

本研究では, 分子動力学法を用いて運動する粒子の温度変化の視覚化を目的とし, 温度による色変化や 3 次元表現, 視点の切り替えによるシミュレーションの見やすさを追求した.

1.2 Maya

Mayaは, オープンアーキテクチャを基盤とした強力な統合型の3Dモデリング, アニメーション, レンダリングソリューションである. 多くのフィルムやビデオアーティスト, ゲーム開発者, マルチメディアデザイナー,3DCGに関わるSOHOデザイナーなどが使用しており, 自動車メーカーなどの製造業では, デザイン的な要素の強い部品の設計に使用している.

このように,Mayaはグラフィック性に優れており, 自由度も高い. また, アニメーションの作成も可能なので, 難解な物理現象も容易に理解することができる.

1.3 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足する. 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できる.

1.3.1 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 $\boldsymbol{r}_i$, 粒子 i に作用する力を $\boldsymbol{f}_i$ とすれば, ニュートンの運動方程式は次のように書ける.

$$m\frac{d^2\boldsymbol{r}_i}{dt^2} = \boldsymbol{f}_i, \quad i = 1, 2, \cdots, N \tag{1.1}$$

速度ベクトル v_i は位置の微分から,

$$\boldsymbol{v}_i = \frac{d\boldsymbol{r}_i}{dt} \tag{1.2}$$

もし, 外力が作用しなければ, 系の運動エネルギーや運動量および角運動量が保存される. しかし, シミュレーションでは一般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.

1.3.2 Verlet のアルゴリズム

Verlet 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という特徴があり, そのために速度スケーリング法が適用できないという性質がある.

Verlet の方法では式 (??) 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 $\boldsymbol{r}_i(t \pm h)$ をテーラー級数に展開し, 式 (??) と $d\boldsymbol{r}_i/dt = \dot{\boldsymbol{r}}_i$ を用いると,

$$\boldsymbol{r}_i(t+h) = \boldsymbol{r}_i(t) + h\dot{\boldsymbol{r}}_i(t) + \frac{h^2}{2}\frac{\boldsymbol{f}_i(t)}{m} + O(h^3) \tag{1.3}$$

$$\boldsymbol{r}_i(t-h) = \boldsymbol{r}_i(t) - h\dot{\boldsymbol{r}}_i(t) + \frac{h^2}{2}\frac{\boldsymbol{f}_i(t)}{m} + O(h^3) \tag{1.4}$$

を得る. 両式の和と差をつくると,

$$\boldsymbol{r}_i(t+h) + \boldsymbol{r}_i(t-h) = 2\boldsymbol{r}_i(t) + h^2\frac{\boldsymbol{f}_i(t)}{m} + O(h^4) \tag{1.5}$$

$$\boldsymbol{r}_i(t+h) - \boldsymbol{r}_i(t-h) = 2h\dot{\boldsymbol{r}}_i(t) + O(h^2) \tag{1.6}$$

これより時刻 $t+h$ における位置と t における速度は

$$\boldsymbol{r}_i(t+h) = 2\boldsymbol{r}_i(t) - \boldsymbol{r}_i(t-h) + h^2\frac{\boldsymbol{f}_i(t)}{m} + O(h^4) \tag{1.7}$$

$$\dot{\boldsymbol{r}}_i(t) = \frac{1}{2h}[\boldsymbol{r}_i(t+h) - \boldsymbol{r}_i(t-h)] + O(h^2) \tag{1.8}$$

で与えられる. 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 $\boldsymbol{r}_i(h)$ は式 (??) から求まる. これと $\boldsymbol{r}_i(0)$ から $\boldsymbol{r}_i(2h)$ を計算し, 式 (??) より速度 $\dot{\boldsymbol{r}}_i(h)$ が得られる.

差分式 (??) の右辺第 3 項の絶対値は第 1 , 2 項に比べて小さいのでこの式から直接 $\boldsymbol{r}_i(t+h)$ を計算しようとすると第 3 項の加算において桁落ちが起こり, 累積誤差が大きくなっていく. これを避けるには,

$$\Delta \boldsymbol{r}_i(t) = \boldsymbol{r}_i(t) - \boldsymbol{r}_i(t-h) \tag{1.9}$$

をつくっておき,

$$\begin{aligned} \Delta \boldsymbol{r}_i(t+h) &= \Delta \boldsymbol{r}_i(t) + h^2 \frac{\boldsymbol{f}_i(t)}{m} \\ \boldsymbol{r}_i(t+h) &= \boldsymbol{r}_i(t) + \Delta \boldsymbol{r}_i(t+h) \end{aligned} \tag{1.10}$$

の 2 打案会に分けて計算する.

この差分式は時間反転（$h \to -h$）に対して対称であり, 速度の符号を変えると系は描いた軌道を逆向きにたどっていく.

1.4 シミュレーション技法

1.4.1 粒子の初期配置と初期速度

実際のシミュレーションは有限のシミュレーション領域で行われる. シミュレーション領域としては, 立方体や直方体に取ることが多い. 立方体のシミュレーション領域の場合, 最密充填格子の一つである面心立方格子状に配置するのが通常である. 直方体の場合も類似の格子状に配置することができる. このような初期位置は気体領域だけでなく, 液体や固体領域の初期状態としても用いることができる. なお, 個体の結晶構造を扱う場合は注意を要し, その結晶構造が取る格子状に格子を配置する必要がある.

以上のように規則的に配置した粒子の初期状態は, シミュレーションの進行とともに, 非常に速やかに熱力学的平衡状態へと推移する.

熱力学的平衡状態にある系の場合, 粒子の速度はマクスウェル分布となる. したがって, 一様乱数列を用いて任意の粒子の速度 $\boldsymbol{v}_i = (v_{ix}, v_{iy}, v_{iz})$ を次のように得る.

$$\left.\begin{aligned} v_{ix} &= (-2\frac{kT}{m}\ln R_j)^{1/2}\cos 2\pi R_{j+1} \\ v_{iy} &= (-2\frac{kT}{m}\ln R_{j+2})^{1/2}\cos 2\pi R_{j+3} \\ v_{iz} &= (-2\frac{kT}{m}\ln R_{j+4})^{1/2}\cos 2\pi R_{j+5} \end{aligned}\right\} \tag{1.11}$$

ここに R_j は一様乱数列から取り出した乱数である. このようにして全粒子の速度を設定すれば, 結果的に粒子の速度はマクスウェル分布に近い分布を与えるが, 粒子数 N が有限なので厳密に等しくなることはない. また, 全粒子の運動量の和すなわち系の運動量はほぼゼロにはなっているが, 十分な精度でゼロになっているとは限らない.

1.4.2 平衡化

分子動力学シミュレーションの場合, 粒子の初期配置と初期速度を与えなければならない. たとえ速度分布を乱数を用いてマクスウェル分布に設定しても, 初期配置とも兼ね合いで, 時間ステップとともに所望の温度から大きくはずれてしまうことがある. したがって, シミュレーションの入る前に, 所望の設定温度や系の運動量がゼロになるよう修正する作業が必要である. これが平衡化の作業である.

ニュートンの運動方程式を用いる分子動力学シミュレーションの場合, 運動エネルギー, すなわち, 瞬間温度は一定でなく時間ステップ毎に変化するので, 各時間ステップで設定温度になるように修正しても意味はない. すなわち, 平均温度が設定温度になるように修正しなければならない.

ある時間間隔 t_s で算出した粒子の平均速度が $\bar{\boldsymbol{v}}$ であるとすると, 系内の全ての粒子, 例えば, 粒子 i の速度 $\boldsymbol{v}_i$ を次式で示すような $\boldsymbol{v'}_i$ で置き換えることにより,

$$\boldsymbol{v'}_i = \boldsymbol{v}_i - \bar{\boldsymbol{v}} \tag{1.12}$$

粒子全体の運動量, すなわち, 系の運動量はゼロとなり, 静止系が得られる.

系が設定温度 T を与えるようにするには, さらに, 次のような速度 $\boldsymbol{v''}_i (i = 1, 2, \cdots, N)$ に置き換える必要がある.

$$\boldsymbol{v''}_i = c_0 \boldsymbol{v'}_i \tag{1.13}$$

ただし, 係数 c_0 は

$$T = \frac{2}{3}N \cdot \frac{1}{k}\langle K \rangle \tag{1.14}$$

を用いて次のように得る.

$$c_0 = \sqrt{3N_s kT/m \sum_{j=1}^{N_s} \boldsymbol{v'}_j^2} \tag{1.15}$$

ここに,N_s は時間間隔 t_s において平均速度を算出する際対象となった粒子数,$\boldsymbol{v'}_j$ はサンプリングされた速度 $\boldsymbol{v}_j$ を式 (??) に従って補正した速度である.

1.4.3 周期境界条件

シミュレーションは有限のシミュレーション領域に対してなされるので, 外部境界条件を設定しなければならない. 周期境界条件は, 気体・液体・個体の区別なく, 全ての状態に適用できる優れた境界条件である.

シミュレーション領域としては, 通常立方体もしくは直方体がよく用いられる. 図??は 2 次元の場合の周期境界条件の概念を示したものである. 中央のセルが対象となる系であり, まわりのセルはその基本セルを複写して作成した仮想のセルである. したがって, ある粒子が境界を通ってシミュレーション領域から流出する場合, 反対側の境界面を通ってそのまま流入することを意味する. さらに, 境界付近の粒子は, 基本セル内の実際の粒子 (実粒子) と複写して作ったセル内の仮想粒子との相互作用を同時に考慮しなければならない. 故に任意の粒子と相互作用する粒子を考える場合, ある実粒子とその複写である実質的に同一な仮想粒子との相互作用を考慮しなければならないことになる. しかし, 最近接像の方法を用いるとどちらか一方の近い方の粒子との相互作用を考慮するだけでよくなる.

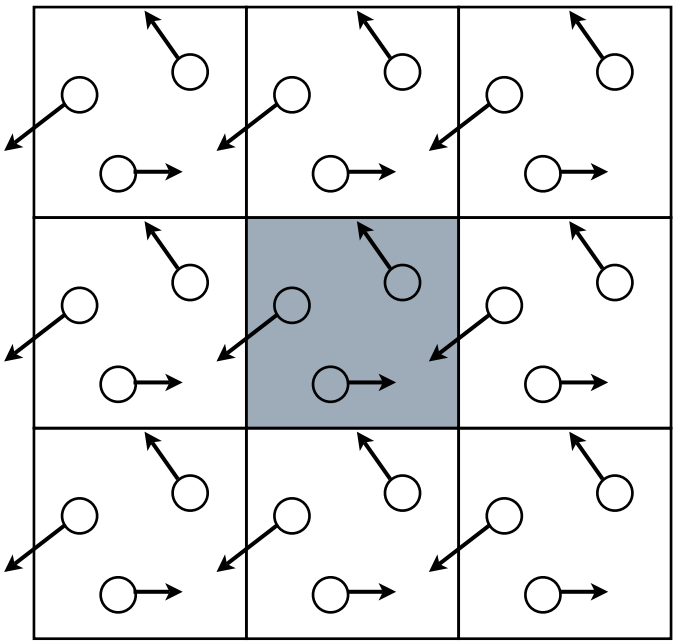


図 1.1: 周期境界条件.

1.4.4 時刻 $t+h$ における位置の算出

シミュレーションでは, 時刻 $t-h$ と時刻 t における粒子配置から時刻 $t+h$ の配置を求める. すなわち,$(x(t-h), y(t-h))$ と $(x(t), y(t))$ が既知であり,$(x(t+h), y(t+h))$ が未知である.$(x(t+h), y(t+h))$ は単純に,

$$x(t+h) = 2x(t) - x(t-h) + \frac{h^2}{m}\boldsymbol{f}_x(t)$$
$$y(t+h) = 2y(t) - y(t-h) + \frac{h^2}{m}\boldsymbol{f}_y(t)$$

のように求めることができる. 次に,

$$(x(t), y(t)) \to (x(t-h), y(t-h))$$

$$(x(t-h), y(t-h)) \to (x(t), y(t))$$

のように置き換えて, 新たに $(x(t+h), y(t+h))$ を算出する. 全粒子についてこの手順を繰り返すことによって, 時間を追って粒子の位置を計算することができる. すなわち, 粒子運動の軌跡を算出することができる.

1.5 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 粒子相互作用の和

$$U = u(r_{12}) + u(r_{13}) + \ldots + u(r_{23}) + \ldots = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u(r_{ij}) \qquad (1.16)$$

となり, ここで $u(r_{ij})$ は粒子 i,j 間の距離 r_{ij} のみに依存する. アルゴンのような単純液体については粒子間相互作用として式 (**??**) の形が適用できる.

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的な計算により第 1 原理から構成することができるものの, 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 (Pauli) の排他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れけばならず, 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になる. 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 (van der Waals) 力と呼ばれている.

最もよく使われる $u(r)$ の形の 1 つはレナード-ジョーンズ (Lennard-Jones)・ポテンシャル

$$u(r) = 4\epsilon \left[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6} \right] \qquad (1.17)$$

である. 図??にレナード-ジョーンズ・ポテンシャルを示す.

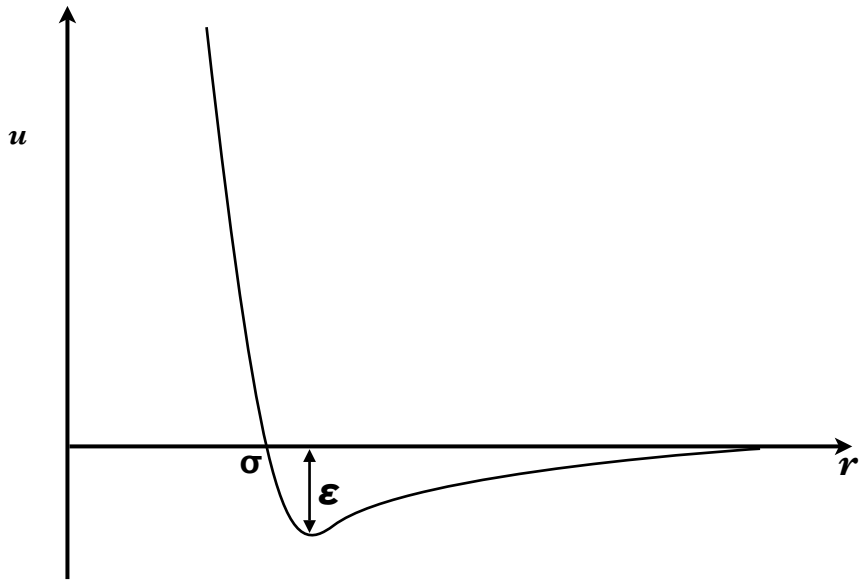


図 1.2: レナード-ジョーンズ・ポテンシャル.

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,$r > 3\sigma$ で $u(r)$ 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り, 極小は $r = 2^{1/6}\sigma$ 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 (??) より分子 i に加わる力 $\boldsymbol{F}_i$ は,

$$\boldsymbol{F}_i = -\frac{\partial U}{\partial \boldsymbol{r}_i} = -\sum_j^N \frac{\partial u(r_{ij})}{\partial \boldsymbol{r}_i} \tag{1.18}$$

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.

$$\frac{\partial r_{ij}}{\partial \boldsymbol{r}_i} = -\frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{1.19}$$

この関係式を用いると,$\boldsymbol{F}_i$ は

$$\boldsymbol{F}_i = -\sum_j^N \frac{\partial u(r_{ij})}{\partial r_{ij}} \cdot \frac{\partial r_{ij}}{\partial \boldsymbol{r}_i} = \sum_j^N \frac{\partial u(r_{ij})}{\partial r_{ij}} \cdot \frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{1.20}$$

となり,r_{ij} に対する偏微分は,

$$\frac{du(r)}{dr} = -24\frac{\epsilon}{\sigma} \cdot \left\{ 2 \cdot (\frac{\sigma}{r})^{13} - (\frac{\sigma}{r})^7 \right\} \tag{1.21}$$

となる.

1.6 温度

温度は粒子の熱運動と関係し, 熱速度, すなわち, 粒子の速度から平均流速を引いた速度で定義される. 今回は系が静止しているものと仮定しているので, 粒子の速度そのものが熱速度となる. 温度 T が与えられた系の場合, 粒子の速度はマクスウェル分布に従った分布となる.

運動エネルギー $K(\boldsymbol{p})$ の集団平均を求めると次のようになる.

$$\langle K(p) \rangle = \left\langle \frac{1}{2m} \sum_{i=1}^N \boldsymbol{p}_i^2 \right\rangle = 3N\frac{kT}{2} \tag{1.22}$$

ただし,$\boldsymbol{p}$は運動量で,$\boldsymbol{p}_1$,$\boldsymbol{p}_2$,$\cdots$,$\boldsymbol{p}_N$ をまとめて表したものである. 小正準集団のように温度が変動する場合, 温度 T は式 (**??**) で定義される.

式 (**??**) より, 運動エネルギー K は

$$K = \frac{3}{2}NkT = \frac{1}{2}mv_2 \tag{1.23}$$

と表すことができ,

$$T = \frac{1}{3}\cdot\frac{1}{kN}mv_2 \tag{1.24}$$

と変形できる.

第2章　手法

2.1　Rubyによるプログラムの作成

Ruby はオブジェクト指向スクリプト言語である. オブジェクト指向とは, プログラムの操作手順よりも操作の対象に重点を置く考えで, プログラムを書いて実行する時間が比較的短く済むという特徴がある. Ruby は変数に型を持たない動的片付けやコンパイルが不要で, 書いてすぐ動かせるインタプリタ方式の仕組みを持ち,Emacs をスクリプトエディタとして使用することができる. Emacs は高機能でフォントが色分けされている等, カスタマイズ性の高いスクリーンエディタである.

数値計算や MEL を出力するコードは Ruby を用いた.

2.1.1　初期条件の設定

粒子の初期配置を指定する. 全粒子に 1〜48 の番号がつけられ,x 軸,y 軸,z 軸それぞれに数値が順に入力される. 運動する粒子は 1〜8 の番号になる. これにより, 全粒子が指定した各座標に配置される. speed() は運動する粒子 8 つにのみ設定し, 速度計算に用いられ, 別に作成したプログラムから毎回ランダムに数値が呼び出される.
(speed() に関しては付録を参照)

```
$init.push [1,[1,1,1],[speed(),speed(),speed()],[2,3,5,9,21,28]]
$init.push [2,[2,1,1],[speed(),speed(),speed()],[1,4,6,10,22,23]]

$init.push [47,[0,2,3]]
$init.push [48,[0,1,3]]
```

```
for i in 0..47
  $x[i] = $init[i][1][0]
  $y[i] = $init[i][1][1]
  $z[i] = $init[i][1][2]
 end
```

同じく初期速度の設定をする. こちらの場合は運動する 8 つの粒子にのみ設定を行う. vx,vy,vz には speed() からランダムな数値が入り, 初期速度となる. ax,ay,az は加速度を表しているが, 初期の状態では常に 0 を指定しておく.

```
for i in 0..7
  $vx[i] = $init[i][2][0]
  $vy[i] = $init[i][2][1]
  $vz[i] = $init[i][2][2]
  $ax[i] = 0
  $ay[i] = 0
  $az[i] = 0
end
```

以上が初期配置と初期速度の設定となる.

2.1.2 レナード-ジョーンズ・ポテンシャルの計算

??にある分子間ポテンシャルの計算を行い, ポテンシャルに生じる力を算出する.

```
r2 = $dx*$dx+$dy*$dy+$dz*$dz
rm2 = 1/r2
rm6 = rm2*rm2*rm2
kk = 24*rm6*(2*rm6-1)*rm2          ←式 (1.21) の計算
fxij = kk*$dx
fyij = kk*$dy
fzij = kk*$dz
$dx = fxij
$dy = fyij
$dz = fzij
```

ポテンシャルに生じる力kkから,x,y,zそれぞれにかかる力を算出し,dx,dy,dzに入れた. また, これらの計算をforce1とし,Verletのアルゴリズムで使用する.

2.1.3 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より呼び出したdx,dy,dzを代入し, 新しくax,ay,azを求める. これらの計算はニュートンの第3法則による.

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

以上をforce2とし, 以下のプログラムで使用する.

以下のプログラムをverlet とし, 粒子の位置を算出する.

```
$x[i] = $x[i]+$vx[i]*$dt+0.5*$ax[i]*$dt2
$y[i] = $y[i]+$vy[i]*$dt+0.5*$ay[i]*$dt2
$z[i] = $z[i]+$vz[i]*$dt+0.5*$az[i]*$dt2
$vx[i] = $vx[i]+0.5*$ax[i]*$dt
$vy[i] = $vy[i]+0.5*$ay[i]*$dt
$vz[i] = $vz[i]+0.5*$az[i]*$dt
```

ここで算出されたx,y,zが粒子の座標になり, 時間きざみで表示される. さらに古い加速度を用いて速度の更新も行っている.

つぎはforce2から新しい加速度を呼び出し, 速度の更新を行う.

```
force2()
    $vx[i] = $vx[i]+0.5*$ax[i]*$dt
    $vy[i] = $vy[i]+0.5*$ay[i]*$dt
    $vz[i] = $vz[i]+0.5*$az[i]*$dt
```

これを時間きざみで行うことで, 粒子の位置を時間経過とともに算出していく.

2.1.4 温度計算

温度Tは式 (??) を用いて計算する.

式 (??) のkはボルツマン定数,Nはアボガドロ定数を示し, 以下のプログラムではk,nとする.rは気体定数で, ボルツマン定数とアボガドロ定数の積で表される.

また, 速度vはverletで算出されたものを用い,$m = 1$とした.

```
k = 1.13806503*10**-23      ←ボルツマン定数
n = 6.0221415*10**23        ←アボガドロ定数
r = k*n                     ←気体定数
$v[i] = sqrt($vx[i]**2+$vy[i]**2+$vz[i]**2)   ←速度
$t[i] = 1/(3*r)*$v[i]*100   ←温度× 100
```

以上により求められた温度tを使い, 温度による色変化を付ける. しかし, このままでは温度差が小さく, 温度による分岐を作成しにくい. そこで温度tを100倍し, 温度差を大きく取っている.

2.2 MELによるプログラムの作成

MEL(Maya Embedded Language) とはMayaで使用できるスクリプト言語である.MayaのGUI(Graphics User Interface) はほとんどMELによって作られていて,Mayaの機能をMELによってコントロールしたり, カスタマイズや拡張などを自由に行うことができる. MELの文法はC言語とUNIXのシェルスクリプトを組み合わせ, 少しC＋＋言語の文法を足したようなものになっている.

2.2.1 カメラの設定

Mayaではカメラを作成することで, レンダリングの際に自由に視点を変えることができる. カメラは3種を選択することができるが, 今回は注視点付きのカメラを作成した. 注視点付きのカメラは, 注視点をフレームの中心に明確に置くことができるため, フレーミングを決めやすいという利点がある. アニメーションの設定時にはカメラ本体と注視点を個別に取り扱うことが可能だが, カメラと注視点をグループ化することで, 通常のカメラと同様に扱うこともできる.

まずは注視点付きカメラを作成した.

```
camera; objectMoveCommand; cameraMakeNode 2 "";
```

次に注視点とカメラの位置を設定する. camera1aimで注視点の座標を,camera1でカメラの座標を指定する.

```
setAttr "camera1_aim.translateX" 1.5;
setAttr "camera1_aim.translateY" 1.5;
setAttr "camera1_aim.translateZ" 1.5;
setAttr "camera1.translateX" 1.5;
setAttr "camera1.translateY" 3;
setAttr "camera1.translateZ" 8;
```

カメラを回転させて視点を切り替えるために,camera1とcamera1aimをグループ化し,group1とした.

```
group; xform -os -piv 0 0 0;
```

group1 は後のメインのプログラムで使われる. このままでは座標 (0,0,0) を中心にカメラが回転するので, 座標 (1.5,1.5,1.5) を中心に回転するよう以下のコマンドで設定する必要がある.

```
select -r group1.scalePivot group1.rotatePivot ;
move -r 1.5 1.5 1.5 ;
```

2.2.2 ライトの設定

ライトを作成し, 粒子に光を当ててレンダリングを行った際の見え方を調節する. 今回はディレクショナルライトを作成した. ディレクショナルライトは平行な無限遠からの光線で, ライトの角度のみ反映され, 位置やスケールは影響しない最も単純なライトである. 光線の角度は一定で減衰がないためレンダリングにかかる時間が短いという利点がある.

ライトの作成には以下の MEL を使用する.

```
defaultDirectionalLight(1, 1,1,1, "0", 0,0,0, 0);
```

括弧の中の数値は, 順に強度, カラー, シャドウの投影, シャドウカラー, インタラクティブの配置を設定している. 強度は最高値の 1, カラーは 1,1,1 で白色, シャドウの投影は 0 でなし, シャドウカラーは 0,0,0 で黒色, インタラクティブの配置は 0 でなし, としている.

次に, 作成したライトの配置を設定する.translateX で x 座標を指定する. 以下のコマンドでは x 座標を 10 に設定している. 同じように y 座標を指定する場合は translateY,z 座標を指定する場合は translateZ を用いる.

```
setAttr "directionalLight1.translateX" 10;
```

ライトの向きを変える場合には,rotate を使用する. 以下のコマンドでは y 軸を中心に 90 °回転するよう設定している. ライトの当たる角度によってレンダリングした際の見え方が変化するので, ここで調節する.x 軸回転にする場合は rotateX,z 軸回転にする場合は rotateZ を用いる.

```
setAttr "directionalLight1.rotateY" 90;
```

均等に光が当たるようにライトを４つ作成し, 向きを回転させ, 四方向から光が当たるように設定する.

2.2.3 粒子となるNURBSの作成および色の設定

粒子には球状NURBSのsphereを使用した. scaleで球の大きさを調節し,moveで座標移動ができる. 今回は完全な球体を使用するため,scaleを全てを0,3に統一した. また, コマンドを繰り返し実行するので, プロシージャの形式にしておくと便利である.

```
global proc md(float $x, float $y, float $z)
{
sphere;
scale 0.3 0.3 0.3;
move $x $y $z;
}
```

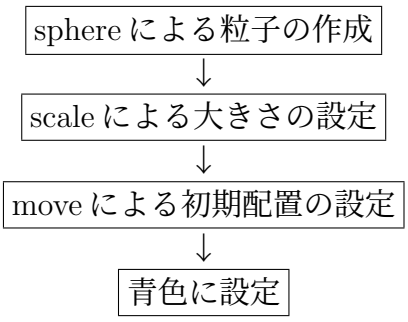
以下のプログラムにはRuby,MELの両方を用いており, 上のコマンドのmoveの座標をここで設定している. 実際のプログラムではここで繰り返しを用いて, 全粒子の配置を設定している.x[i],y[i],z[i]には座標が入る.

```
md($x[i].to_s , $y[i].to_s , $z[i].to_s );
```

続いて初期配色の設定を行った. ここでは初期の状態, つまり速度が0の時に青色になるよう設定した.lambert1はsphereが作成されると自動的に設定されるノードなので, 新しくノードを作成する必要はない. 以下のコマンドで0 0 1というのは青色を指定している.

```
setAttr lambert1.color -type double3  0 0 1 ;
```

??での流れをまとめると,



というプログラムを繰り返すことで粒子を作成した.

2.2.4 温度による色変化の設定

色を変化させるには, 変化する 8 つの sphere それぞれにノードを作成する必要がある. ノードは以下のコマンドで作成する.

```
shadingNode -asShader lambert;
sets -renderable true -noSurfaceShader true -empty -name lambert2SG;
connectAttr -f lambert2.outColor lambert2SG.surfaceShader;
setAttr "lambert2.color" -type double3 0 0 1 ;
```

今回は光沢のない質感の lambert を使用したが, 他のマテリアルを使用する場合は lambert のコマンドを変更するだけで違うマテリアルを作成できる. また, ノードの表面の色をキーフレームで設定し変化させていくために, 2 行目と 3 行目のコマンドが必要になり, 4 行目は初期の色を青色に設定している. 以上のプログラムを繰り返し,lambert2〜lambert9 のノードを作成する.

続いて作成したノードを sphere に適用させていく.

```
select -r nurbsSphere1;
sets -e -forceElement lambert2SG;
```

これを 8 つ全ての sphere に行い,

nurbsSphere1 … lambert2
nurbsSphere2 … lambert3
nurbsSphere3 … lambert4
nurbsSphere4 … lambert5
nurbsSphere5 … lambert6
nurbsSphere6 … lambert7
nurbsSphere7 … lambert8
nurbsSphere8 … lambert9

と設定した.

2.2.5　キーフレームの設定

オブジェクトのアトリビュートの記録をキーと呼び, 時間軸上に指定されたキーをキーフレームと呼ぶ. ２つ以上のキーフレームを作るとオブジェクトはアニメーションする. キーフレームを編集することでアニメーションのタイミングを調節できる.

基本的なキーフレームアニメーションの手順は以下のようになっている.

1. currentTime で現在の時間を設定.
2. オブジェクトのアトリビュートを変更.
3. setKeyframe でキーフレームを決定.
4. 手順１～３を必要なフレーム数だけ繰り返す.

まず, スクリプトに使用する names[ ] に,sphere の１～８とカメラとカメラの焦点をグループ化した group1 を入れる.

```
select -r nurbsSphere1 nurbsSphere2 nurbsSphere3 nurbsSphere4
  nurbsSphere5 nurbsSphere6 nurbsSphere7 nurbsSphere8 group1;
string $names[] = `ls -sl`;
```

これを行うことで,

names[0] → nurbsSphere1
names[1] → nurbsSphere2
names[2] → nurbsSphere3
names[3] → nurbsSphere4
names[4] → nurbsSphere5
names[5] → nurbsSphere6
names[6] → nurbsSphere7
names[7] → nurbsSphere8
names[8] → group1

と対応するようになる.

今回のアニメーションには０～１００のフレームを使用した.
フレーム０とフレーム１００にはカメラが回転するスクリプトをつける.

```
currentTime 0;
setAttr($names[8] + ".ry") 0;
setKeyframe -at "ry" $names[8];

currentTime 100;
setAttr($names[8] + ".ry") 359;
setKeyframe -at "ry" $names[8];
```

これで, フレーム０からフレーム１００の間に names[8](group1), つまりカメラが y 軸を中心に 360 °回転するよう設定できた. フレーム１～９９の間は自動的に移動してくれるので, わざわざ設定する必要はない.

続いて,names[0]～[7](nurbsSphere1～nurbsSphere8), つまり運動している粒子の位置をキーフレームで設定する.

```
setAttr($names[0] + ".tx")  $x;
setAttr($names[0] + ".ty")  $y;
setAttr($names[0] + ".tz")  $z;
setKeyframe -at "tx" $names[0];
setKeyframe -at "ty" $names[0];
setKeyframe -at "tz" $names[0];
```

setAttr で names[0](nurbsSphere1) のアトリビュートに x,y,z の座標を設定する. この座標には,Verlet アルゴリズムで算出されたそれぞれの座標が入る. setKeyframe は,setAttr で設定されたアトリビュートにキーフレームを設定している. この作業をフレーム０～１００の間に, 運動する８つの粒子それぞれに設定することで, 分子動力学法によって算出された座標に粒子が移動するというアニメーションを作成することができる.

第3章　結果

図 (??) は Maple により視覚化された結果, 図 (??) は Maya により視覚化された結果を正面から見たものである. これらを比較すると, 2 つの相違点が分かる. まず,Maya の方が見た目がよく, 色や質感の表現に優れている. また, 温度変化の視覚化が可能になったことが大きな違いである. 青色は速度が 0 の状態を示し, 速度が上がるにつれて温度も上昇し, 赤色へと変化している. アニメーションで見ると, 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.

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, アニメーションとしては簡易なものになってしまう. また, 描画された結果は正面からしか見ることができず, 視点の角度を変えるにはプログラムを書き直さなければならない. 一方,Maya は設定さえしておけば自由な角度でのアニメーションが可能な上, カメラ視点以外でも front,side,top と簡単に視点の切り替えができる.

Maya でのプログラム読み込みの際, 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. この場合,Maya のスクリプトエディタで直接直す必要があるが, 読み込めない行を表示するようにしておけばバグを探す手間が省ける.

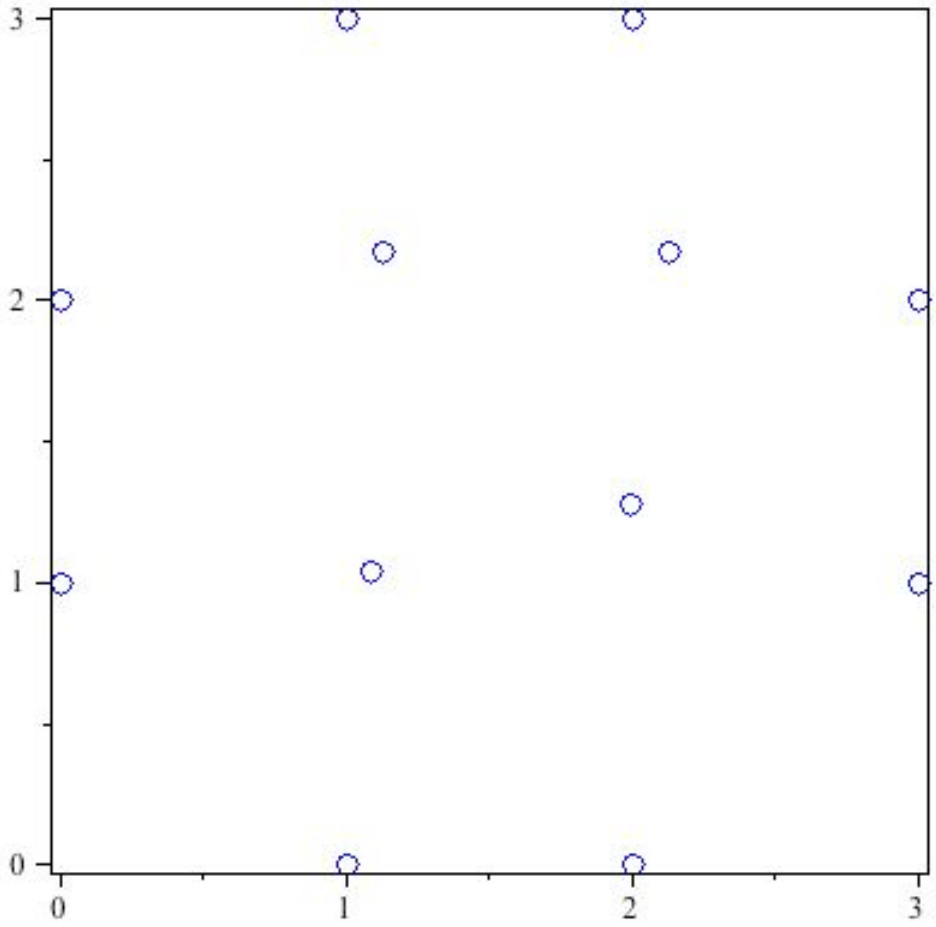


図 3.1: Maple により視覚化された結果.

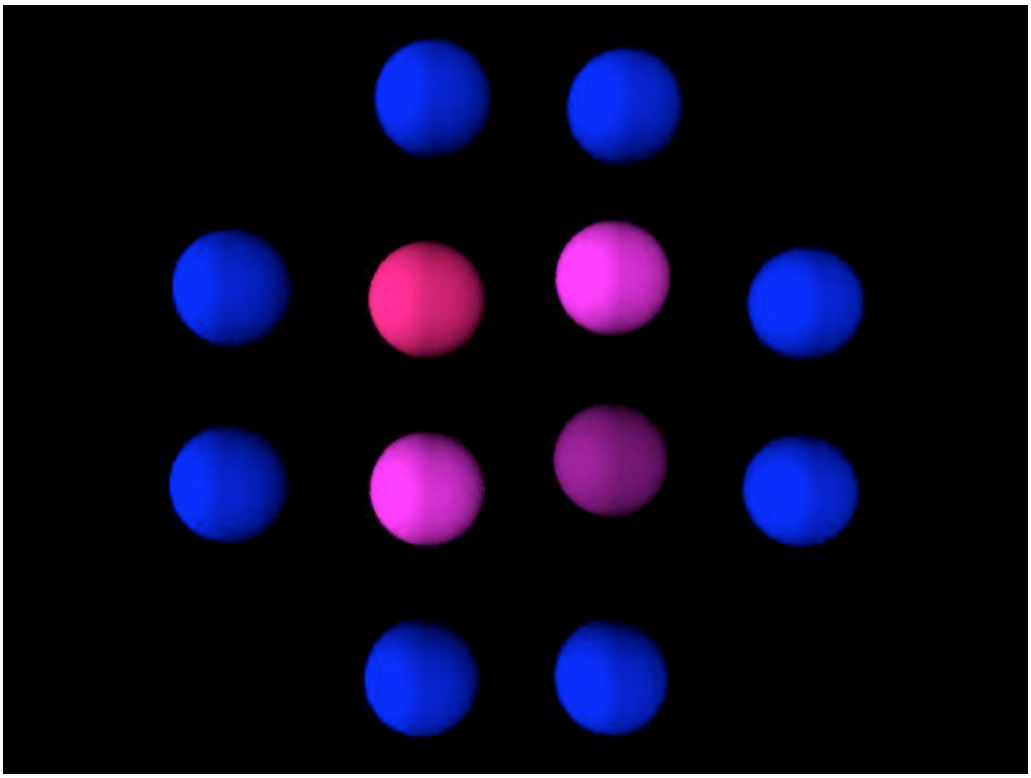


図 3.2: Maya により視覚化された結果.

第4章　総括

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, ご協力を頂き, 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.

付 録A　プログラム

A.1　Emacs

camera

```
def camera()
  print 'camera ; objectMoveCommand; cameraMakeNode 2 "";'
  + "\n"
  print 'setAttr "camera1_aim.translateX" 1.5;' + "\n"
  print 'setAttr "camera1_aim.translateY" 1.5;' + "\n"
  print 'setAttr "camera1_aim.translateZ" 1.5;' + "\n"
  print 'setAttr "camera1.translateX" 1.5;' + "\n"
  print 'setAttr "camera1.translateY" 3;' + "\n"
  print 'setAttr "camera1.translateZ" 8;' + "\n"
  print 'select -r camera1 ;' + "\n"
  print 'group; xform -os -piv 0 0 0;' + "\n"
  print 'select -r group1.scalePivot group1.rotatePivot ;'
  + "\n"
  print 'move -r 1.5 1.5 1.5 ;' + "\n"
end
```

light

```
def light()
 print 'defaultDirectionalLight(1, 1,1,1, "0", 0,0,0, 0);'
  + "\n"
 print 'setAttr "directionalLight1.translateX" 10;' + "\n"
 print 'setAttr "directionalLight1.rotateY" 90;' + "\n"
 print 'defaultDirectionalLight(1, 1,1,1, "0", 0,0,0, 0);'
  + "\n"
```

light

```
 print 'setAttr "directionalLight2.translateX" -10;' + "\n"
 print 'setAttr "directionalLight2.rotateY" -90;' + "\n"
 print 'defaultDirectionalLight(1, 1,1,1, "0", 0,0,0, 0);'
  + "\n"
 print 'setAttr "directionalLight3.translateZ" 10;' + "\n"
 print 'defaultDirectionalLight(1, 1,1,1, "0", 0,0,0, 0);'
  + "\n"
 print 'setAttr "directionalLight4.translateZ" -10;' + "\n"
 print 'setAttr "directionalLight4.rotateY" 180;' + "\n"
end
```

color

```
Def color()
  for k in 2..9
    print 'shadingNode -asShader lambert;' + "\n"
    print 'sets -renderable true -noSurfaceShader true
     -empty -name lambert' + k.to_s + 'SG;' + "\n"
    print 'connectAttr -f lambert' + k.to_s + '.outColor
     lambert' + k.to_s + 'SG.surfaceShader;' + "\n"
    print 'setAttr "lambert' +  k.to_s + '.color" -type
     double3 0 0 1 ;' + "\n"
  end
end
```

speed

```
def speed()
  3*rand
end
```

force1

```
def force1()
  R2 = $dx*$dx+$dy*$dy+$dz*$dz
  rm2 = 1/r2
  rm6 = rm2*rm2*rm2
  kk = 24*rm6*(2*rm6-1)*rm2
  fxij = kk*$dx
  fyij = kk*$dy
  fzij = kk*$dz
  $dx = fxij
  $dy = fyij
  $dz = fzij
end
```

force2

```
def force2()
  for i in 0..7
    $ax[i] = 0
    $ay[i] = 0
    $az[i] = 0
  end
  for i in 0..7
    for j in 0..5
      jj = $init[i][3][j]-1
      $dx = $x[i] - $x[jj]
      $dy = $y[i] - $y[jj]
      $dz = $z[i] - $z[jj]
      force1()
      $ax[i] = $ax[i] + $dx
      $ay[i] = $ay[i] + $dy
      $az[i] = $az[i] + $dz
    end
  end
end
```

verlet

```
def verlet()
  for i in 0..7
    $x[i] = $x[i]+$vx[i]*$dt+0.5*$ax[i]*$dt2
    $y[i] = $y[i]+$vy[i]*$dt+0.5*$ay[i]*$dt2
    $z[i] = $z[i]+$vz[i]*$dt+0.5*$az[i]*$dt2
    $vx[i] = $vx[i]+0.5*$ax[i]*$dt
    $vy[i] = $vy[i]+0.5*$ay[i]*$dt
    $vz[i] = $vz[i]+0.5*$az[i]*$dt
  end
  force2()
  for i in 0..7
    $vx[i] = $vx[i]+0.5*$ax[i]*$dt
    $vy[i] = $vy[i]+0.5*$ay[i]*$dt
    $vz[i] = $vz[i]+0.5*$az[i]*$dt
  end
end
```

temperature

```
def temperature()
  verlet()
  for i in 0..7
    k = 1.13806503*10**-23
    n = 6.0221415*10**23
    r = k*n
    $v[i] = sqrt($vx[i]**2+$vy[i]**2)
    $t[i] = 1/(3*r)*$v[i]*100
  end
end
```

main

```
include Math

require 'camera'
require 'light'
require 'color'
require 'speed'
require 'force1'
require 'force2'
require 'verlet'
require 'temple'
$init = []
$init.push [1,[1,1,1],[speed(),speed(),speed()],
[2,3,5,9,21,28]]
$init.push [2,[2,1,1],[speed(),speed(),speed()],
[1,4,6,10,22,23]]
$init.push [3,[1,2,1],[speed(),speed(),speed()],
[1,4,7,11,26,27]]
$init.push [4,[2,2,1],[speed(),speed(),speed()],
[2,3,8,12,24,25]]
$init.push [5,[1,1,2],[speed(),speed(),speed()],
[1,6,7,29,36,37]]
$init.push [6,[2,1,2],[speed(),speed(),speed()],
[2,5,8,30,31,38]]
$init.push [7,[1,2,2],[speed(),speed(),speed()],
[3,5,8,34,35,39]]
$init.push [8,[2,2,2],[speed(),speed(),speed()],
[4,6,7,32,33,40]]
$init.push [9,[1,1,0]]
$init.push [10,[2,1,0]]
$init.push [11,[1,2,0]]
$init.push [12,[2,2,0]]
$init.push [13,[1,0,0]]
$init.push [14,[2,0,0]]
$init.push [15,[3,1,0]]
$init.push [16,[3,2,0]]
```

main

```
$init.push [17,[2,3,0]]
$init.push [18,[1,3,0]]
$init.push [19,[0,2,0]]
$init.push [20,[0,1,0]]
$init.push [21,[1,0,1]]
$init.push [22,[2,0,1]]
$init.push [23,[3,1,1]]
$init.push [24,[3,2,1]]
$init.push [25,[2,3,1]]
$init.push [26,[1,3,1]]
$init.push [27,[0,2,1]]
$init.push [28,[0,1,1]]
$init.push [29,[1,0,2]]
$init.push [30,[2,0,2]]
$init.push [31,[3,1,2]]
$init.push [32,[3,2,2]]
$init.push [33,[2,3,2]]
$init.push [34,[1,3,2]]
$init.push [35,[0,2,2]]
$init.push [36,[0,1,2]]
$init.push [37,[1,1,3]]
$init.push [38,[2,1,3]]
$init.push [39,[1,2,3]]
$init.push [40,[2,2,3]]
$init.push [41,[1,0,3]]
$init.push [42,[2,0,3]]
$init.push [43,[3,1,3]]
$init.push [44,[3,2,3]]
$init.push [45,[2,3,3]]
$init.push [46,[1,3,3]]
$init.push [47,[0,2,3]]
$init.push [48,[0,1,3]]
$x = []
$y = []
$z = []
```

main

```
$ax = []
$ay = []
$az = []
$vx = []
$vy = []
$vz = []
$v = []
$t =[]
$dt = 0.01
$dt2 = $dt*$dt

camera()
light()
color()

print 'global proc md(float $x, float $y, float $z)' + "\n"
print '{' + "\n"
print 'sphere;' + "\n"
print 'scale 0.3 0.3 0.3;' + "\n"
print 'move $x $y $z;' + "\n"
print '}' + "\n"

for i in 0..47
  $x[i] = $init[i][1][0]
  $y[i] = $init[i][1][1]
  $z[i] = $init[i][1][2]
  print 'md(' + $x[i].to_s + ',' + $y[i].to_s + ','
  + $z[i].to_s + ');' + "\n"
  print 'setAttr lambert1.color -type double3  0 0 1 ;'
   + "\n"
end
```

main

```
for i in 0..7
  $vx[i] = $init[i][2][0]
  $vy[i] = $init[i][2][1]
  $vz[i] = $init[i][2][2]
  $ax[i] = 0
  $ay[i] = 0
  $az[i] = 0
end

print "\n"
print 'select -r nurbsSphere1 ;'+ "\n"
print 'sets -e -forceElement lambert2SG;' + "\n"
print 'select -r nurbsSphere2 ;'+ "\n"
print 'sets -e -forceElement lambert3SG;' + "\n"
print 'select -r nurbsSphere3 ;'+ "\n"
print 'sets -e -forceElement lambert4SG;' + "\n"
print 'select -r nurbsSphere4 ;'+ "\n"
print 'sets -e -forceElement lambert5SG;' + "\n"
print 'select -r nurbsSphere5 ;'+ "\n"
print 'sets -e -forceElement lambert6SG;' + "\n"
print 'select -r nurbsSphere6 ;'+ "\n"
print 'sets -e -forceElement lambert7SG;' + "\n"
print 'select -r nurbsSphere7 ;'+ "\n"
print 'sets -e -forceElement lambert8SG;' + "\n"
print 'select -r nurbsSphere8 ;'+ "\n"
print 'sets -e -forceElement lambert9SG;' + "\n"
print 'select -r nurbsSphere1  nurbsSphere2 nurbsSphere3
  nurbsSphere4 nurbsSphere5  nurbsSphere6 nurbsSphere7
  nurbsSphere8  group1 ;'+ "\n"
print 'string $names[] = `ls -sl`;' + "\n"
print "\n"
```

main

```
 verlet()
print 'currentTime 0;' + "\n"
print 'setAttr($names[8] + ".ry") 0;' + "\n"
print 'setKeyframe -at "ry" $names[8];' + "\n"
for j in 0..7
  print 'setAttr($names[' +  j.to_s + '] + ".tx") '
    + $x[j].to_s + ';' + "\n"
  print 'setAttr($names[' +  j.to_s + '] + ".ty") '
    + $y[j].to_s + ';' + "\n"
  print 'setAttr($names[' +  j.to_s + '] + ".tz") '
    + $z[j].to_s + ';' + "\n"
  print 'setKeyframe -at "tx" $names[' +  j.to_s + '];'
    + "\n"
  print 'setKeyframe -at "ty" $names[' +  j.to_s + '];'
    + "\n"
  print 'setKeyframe -at "tz" $names[' +  j.to_s + '];'
    + "\n"
end

for i in 1..99
  print 'currentTime ' +  i.to_s + ';' + "\n"
  for j in 0..7
    verlet()
    print 'setAttr($names[' +  j.to_s + '] + ".tx") '
      + $x[j].to_s + ';' + "\n"
    print 'setAttr($names[' +  j.to_s + '] + ".ty") '
      + $y[j].to_s + ';' + "\n"
    print 'setAttr($names[' +  j.to_s + '] + ".tz") '
      + $z[j].to_s + ';' + "\n"
    print 'setKeyframe -at "tx" $names[' +  j.to_s + '];'
      + "\n"
    print 'setKeyframe -at "ty" $names[' +  j.to_s + '];'
      + "\n"
    print 'setKeyframe -at "tz" $names[' +  j.to_s + '];'
      + "\n"
```

main

```
temple()
if ($t[j] > 20)
  print 'select -r lambert' +  (j+2).to_s + ' ;'+ "\n"
  print 'setAttr "lambert' +  (j+2).to_s + '.color"
        -type double3 0 1 1 ;' + "\n"
  print 'setKeyframe -breakdown 0 -hierarchy none
        -controlPoints 0 -shape 0 {"lambert' +
        (j+2).to_s + '"};' + "\n"
elsif ($t[j] <= 20 && $t[j] >= 17)
  print 'select -r lambert' +  (j+2).to_s + ' ;'+ "\n"
  print 'setAttr "lambert' +  (j+2).to_s + '.color"
        -type double3 0 0.5 1 ;' + "\n"
  print 'setKeyframe -breakdown 0 -hierarchy none
        -controlPoints 0 -shape 0 {"lambert' +
        (j+2).to_s + '"};' + "\n"
elsif ($t[j] <= 17 && $t[j] >= 14)
  print 'select -r lambert' +  (j+2).to_s + ' ;'+ "\n"
  print 'setAttr "lambert' +  (j+2).to_s + '.color"
        -type double3 0 0 1 ;' + "\n"
  print 'setKeyframe -breakdown 0 -hierarchy none
        -controlPoints 0 -shape 0 {"lambert' +
        (j+2).to_s + '"};' + "\n"
elsif ($t[j] <= 14 && $t[j] >= 11)
  print 'select -r lambert' +  (j+2).to_s + ' ;'+ "\n"
  print 'setAttr "lambert' +  (j+2).to_s + '.color"
        -type double3 0.5 0 1 ;' + "\n"
  print 'setKeyframe -breakdown 0 -hierarchy none
        -controlPoints 0 -shape 0 {"lambert' +
        (j+2).to_s + '"};' + "\n"
elsif ($t[j] <= 11 && $t[j] >= 8)
  print 'select -r lambert' +  (j+2).to_s + ' ;'+ "\n"
  print 'setAttr "lambert' +  (j+2).to_s + '.color"
        -type double3 0.5 0 0.5 ;' + "\n"
  print 'setKeyframe -breakdown 0 -hierarchy none
        -controlPoints 0 -shape 0 {"lambert' +
```

main

```
          (j+2).to_s + '"};' + "\n"
    elsif ($t[j] <= 8 && $t[j] >= 5)
      print 'select -r lambert' +  (j+2).to_s + ' ;'+ "\n"
      print 'setAttr "lambert' +  (j+2).to_s + '.color"
            -type double3 1 0 0.5 ;' + "\n"
      print 'setKeyframe -breakdown 0 -hierarchy none
            -controlPoints 0 -shape 0 {"lambert' +
            (j+2).to_s + '"};' + "\n"
    elsif ($t[j] <= 5 && $t[j] >= 2)
      print 'select -r lambert' +  (j+2).to_s + ' ;'+ "\n"
      print 'setAttr "lambert' +  (j+2).to_s + '.color"
            -type double3 1 0 1 ;' + "\n"
      print 'setKeyframe -breakdown 0 -hierarchy none
            -controlPoints 0 -shape 0 {"lambert' +
            (j+2).to_s + '"};' + "\n"
    elsif ($t[j] < 2)
      print 'select -r lambert' +  (j+2).to_s + ' ;'+ "\n"
      print 'setAttr "lambert' +  (j+2).to_s + '.color"
            -type double3 1 0 0 ;' + "\n"
      print 'setKeyframe -breakdown 0 -hierarchy none
            -controlPoints 0 -shape 0 {"lambert' +
            (j+2).to_s + '"};'+ "\n"
    end
  end
end

print 'currentTime 100;' + "\n"
print 'setAttr($names[8] + ".ry") 359;' + "\n"
print 'setKeyframe -at "ry" $names[8];' + "\n"
for j in 0..7
  verlet()
  print 'setAttr($names[' +  j.to_s + '] + ".tx") '
   + $x[j].to_s + ';' + "\n"
```

main

```
  print 'setAttr($names[' +  j.to_s + '] + ".ty") '
   + $y[j].to_s + ';' + "\n"
  print 'setAttr($names[' +  j.to_s + '] + ".tz") '
   + $z[j].to_s + ';' + "\n"
  print 'setKeyframe -at "tx" $names[' +  j.to_s + '];'
   + "\n"
  print 'setKeyframe -at "ty" $names[' +  j.to_s + '];'
   + "\n"
  print 'setKeyframe -at "tz" $names[' +  j.to_s + '];'
   + "\n"
end
```

関連図書

[1] 神山新一, 佐藤 明.『分子動力学シミュレーション』

[2] 北川 浩.『初心者のための分子動力学法』

[3] 上田　.『分子シミュレーション』

[4] Harvey Gould,Jan Tobochnik.『計算物理学入門』

[5] Chris Pine.『はじめてのプログラミング』

[6] 林田宏之, 橋口智仁,konkon, 黒田あや子, 本城なお.『AUTODESK MAYA オフィシャルトレーニングブック』『AUTODESK MAYA ビジュアルリファレンス』

[7] 阿部知弘.『MEL 教科書 -Maya プログラミング入門-』

[8] David Stripinis.『THE MAL COMPANION : MAYA SCRIPTING for 3D ARTISTS』

[9] Mark R wilkins,Chris Kazmier.『MAYA Scripting for MAYA Animators』