mdspass2 改良点について（2016年5月19日）

（最新版は<http://www.cmsm.iis.u-tokyo.ac.jp/code/mdspass2/forWin/> にアップ済）

1. 原子間ボンド表示について

* Set param 🡪 Setup for drawing とクリックすると開くボンド表示設定部分(“Bond”と書かれた枠で囲まれた部分)に機能を追加しました。
* ボンド表示のタイプをLine/Cylinderから選べるようにしました。前者は立体感の無い線で、後者は立体感のある棒で表示します。描画速度は当然後者の方が遅くなりますが原子数が少ない場合には問題ないでしょう。
* Cylinderの場合、色が選択できるようにしてあります。下向き矢印の付いたウインドウを選ぶとリストが表示されます。
* ボンドの太さを変えることができるようにしました。Lineの場合、7で頭打ちとなりそれ以上の値を入れても変わりません。
* なお、この場合のボンドは、Bond lengthで表示されている値（単位はÅ）範囲内にある原子同士を単純につなげているだけです。結合状態を計算しているわけではありません（古典分子動力学ですので）。

1. 特定原子のマーキングについて

* Set param 🡪 Atom color で、原子の着色法を設定するパネルが開きます。ここで、特定の原子3つまでを選択し、好みの色で表示するという機能を付加しました。
* Marked atom 0/1/2 で原子の番号を選択します。原子の番号が分からない場合には、MD viewerウィンドウ上で原子をクリックすれば、コンソール画面上（Windowsではコマンドプロンプトのような黒いウィンドウ）に選択した原子の情報が表示されるので、それを参考にしてください。着色を止めたければ、この値をゼロにしてください。
* 選択した原子にそれぞれ別の色を与えることができます。

1. 圧力の算出法について

* ナノチューブの解析の場合（Set param 🡪 Special settings for CNT 🡪 Corrugation modeをチェックした場合）、Stressでパネルを開くとCNT corrugationの部分に圧力が表示されるようになっていました。この時チューブの最外層の表面積を求めるルーチンが、断面形状＝真円を仮定していたため、断面がつぶれた時に間違った値を吐く、という問題がありました。
* そこで、今回は断面がつぶれた場合にも正しく表面積を出してくれるように改良しました。多層ナノチューブの場合、最外層のチューブのみを検出してその表面積を計算してくれるようにしています。ユーザー側で特に行うべき操作はありません。
* これに伴って、Outermost radiusの表示を廃止しました。

ナノチューブ表面積計算ルーチン（20160519版）

* 1. チューブの中心がｚ軸に沿うと仮定する。中心軸の位置(x,y)を決め、中心軸から極座標表示した原子の座標r()を計算し、メッシュデータとして記憶する。  
     , rをそれぞれ the[i], rad[i] と配列変数に保存する。iはカウンタで、要素の数はnum\_elemとする。この時点ではnum\_elem = 原子数。
  2. the[i], rad[i] をthe () の昇順にソートする。
  3. 最外殻のみを取り出す。iを0から一つずつ増加させていき、要素iとその次の要素に対する半径rの差drad = rad[i] - rad[i-1] を求める。この値が負（かつ絶対値が閾値より大）であれば、最外殻でない原子が見つかったことになるので、その要素を削除する（それより大きな要素に対するデータを一つずつ下げる）。値が正（かつ絶対値が閾値より大）であれば、要素iおよびそれより小さな要素はすべて最外殻でない原子のデータであるので、これを削除する。
  4. これで最外殻の原子の座標r()が  の昇順に得られた。密に詰まりすぎている要素を削除しておく。
  5. 得られたr()を用いて、を計算し、セルのｚ方向長さをかけるとチューブの表面積が得られる。

// Side area calculation for an arbitrary cross section shape

void calc\_cylinder\_side\_area\_arb(double &area)

{

double xx, yy;

double \*rad, \*the, \*dum; // radius, theta, dummy

int \*nod; // order // nod[] (配列) はソートの際に生成される。データを昇順に数えた際にどのような順番になるかをストアする。

int num\_elem = atom.natom; // # of mesh for r(theta)

rad = new double[atom.natom]; // r

the = new double[atom.natom]; // theta

dum = new double[atom.natom]; // dummy

nod = new int[atom.natom]; // order

// calculate center (x,y)

double center\_x, center\_y;

calc\_center\_xy(center\_x, center\_y);

// store r(theta)

for (int i=1; i<=atom.natom; i++) {

xx = atom.rx[i] - center\_x;

yy = atom.ry[i] - center\_y;

rad[i-1] = sqrt(xx\*xx + yy\*yy);

the[i-1] = theta(xx, yy)/M\_PI\*180.0;

}

// sort r(theta) in order of theta

sort(num\_elem, the, nod);

for (int i=0; i<num\_elem; i++) { dum[i] = rad[i]; }

for (int i=0; i<num\_elem; i++) { rad[i] = dum[nod[i]]; }

//for (int i=0; i<num\_elem; i++) { printf("%d %e %e\n",i,the[i],rad[i]); }

// sieve out outermost wall

double threshold = 0.2 \* ang;

for (int i=1; i<num\_elem; i++) {

double drad = rad[i] - rad[i-1];

if (drad > threshold) {

for (int j=i; j<num\_elem; j++) {

rad[j-i] = rad[j]; the[j-i] = the[j];

}

num\_elem -= i; i=0;

} else if (drad < -threshold) {

for (int j=i; j<num\_elem-1; j++) {

rad[j] = rad[j+1]; the[j] = the[j+1];

}

num\_elem--; i--;

}

}

//for (int i=0; i<num\_elem; i++) { printf("%d %e %e\n",i,the[i],rad[i]); }

// remove crowded points

threshold = 0.2;

for (int i=1; i<num\_elem; i++) {

double dthe = the[i] - the[i-1];

if (fabs(dthe) < threshold) {

for (int j=i; j<num\_elem-1; j++) {

rad[j] = rad[j+1]; the[j] = the[j+1];

}

num\_elem--; i--;

}

}

//for (int i=0; i<num\_elem; i++) { printf("%d %e %e\n",i,the[i],rad[i]); }

// calculate path length

double path = 0.0;

for (int i=0; i<num\_elem; i++) {

double dthe;

if (i<num\_elem-1) {

dthe = the[i+1]-the[i];

if (dthe < 0.0) { dthe += 360.0; }

dthe \*= M\_PI/180.0;

path += (rad[i+1]+rad[i])\*dthe/2.0;

} else {

dthe = the[0]-the[i];

if (dthe < 0.0) { dthe += 360.0; }

dthe \*= M\_PI/180.0;

path += (rad[0]+rad[i])\*dthe/2.0;

}

}

area = path\*cell.hmat[2][2]; //(in m^2)

}