

卒業論文

三面図を利用した粒界原子配列の視覚化

関西学院大学 理工学部 情報科学科

1549 成田 大樹

2017 年 3 月

指導教員 西谷 滋人 教授

目次

1	序論	4
2	ソフト開発の手法	7
2.1	MVC モデルの概要と利点	7
2.2	SVG 表示の特徴	8
2.3	rcairo を使用する利点	8
2.4	一般的な原子モデルソフトとの比較	8
3	ソフトの構成と描画	11
3.1	外部データの読み込み部	11
3.1.1	read_pos	11
3.2	原子座標の計算部	12
3.2.1	identical_atoms	12
3.2.2	mk_deleted_atom	13
3.3	粒界原子の描画部	14
3.3.1	draw_backcolor, draw_exes	14
3.3.2	open_circle	14
3.3.3	draw_each_plane, pos_y	14
3.3.4	draw_atoms	16
3.3.5	find_max	16
3.3.6	main_draw	16
3.4	三面図による描画	17
4	結果と考察	19
4.1	各用途に合わせた原子配列の表示結果	19
4.1.1	構造緩和による原子移動の表示	19
4.1.2	削除された原子の識別表示	19
4.1.3	指定した層の白抜き表示	20
4.2	原子構造の改善点	20
4.3	考察と今後の課題	22

5	総括	23
6	参考文献	24
	{{toc}} 概要 (boundary_narita_summary)	

1 序論

本研究の目的は、小傾角を用いたシミュレーションで解明すること。シミュレーション結果では、0°がそれぞれ異なる傾きに称傾角粒界エネルギーの

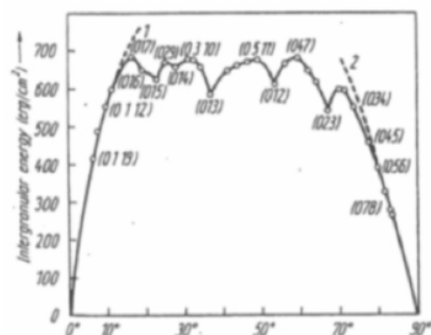


図 1 Hasson らによるシミュレーション結果。

Hasson らによるシミュレーションは、Read-Shockley の理論予測をもとにおこなった。Read-Shockley の式は以下ようになる。

小傾角粒界エネルギーによる 2 つの結果の矛盾を明白にするために、西谷研究室において様々な検証が今までおこなわれてきた。はじめに、第一原理計算ソフト VASP による計算をおこない、その後、原子間ポテンシャルを使ったシミュレーションや Sutton Vitek による粒子モデルの研究を取り組んできた。また、八幡の研究では、経験的原子間ポテンシャルによるシミュレーションをおこない、Read-Shockley の理論予測と同様の結果となった [1]。さらに、岩佐の研究では、最安定な原子配置を探索するために原子の削除操

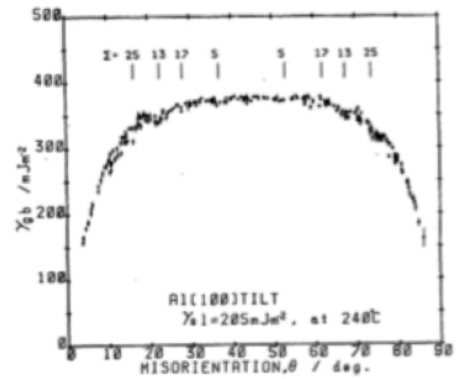


図 2 大槻による実験結果．

作を取り入れ，第一原理計算ソフト VASP を用いて構造緩和し，系全体のエネルギー計算をおこなった．その結果，小傾角粒界エネルギーが予測通りに大槻の結果よりも低いエネルギーの値となり，粒界エネルギーの立ち上がり 0 度付近が最安定の構造になった．しかしながら，粒界がより低い角度になった状態を計算していたため，図 004 のように原子が全体的に傾いた構造になり，全ての構造が最安定であることの検証までには至らなかった．エネルギー計算による誤った構造緩和は，安定構造の原子配置を視覚的に確認しなかったことが原因である [2]．

この失敗の原因を踏まえて，原子配置をより簡易的で視覚化できるプログラムを作成していく．したがって，粒界原子配列を正確で，且つ容易に視覚化できるためのソフトを開発することが本研究の目標である．本研究では小傾角粒界の原子モデルの配列を 2 次元で描画するソフトを開発する．

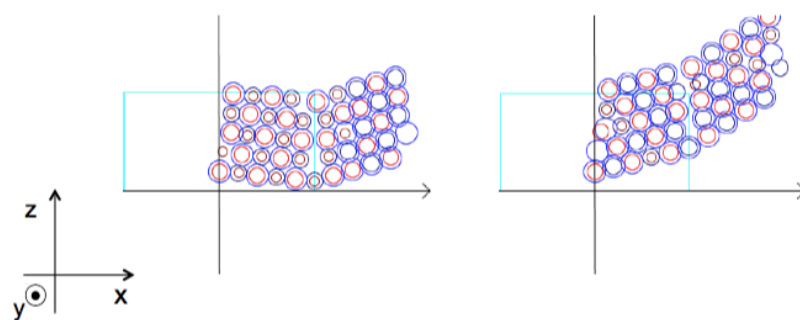


図 3 岩佐による誤った構造緩和の結果

2 ソフト開発の手法

本研究で開発するソフト”viewer”は，小傾角粒界の原子モデルの配列を 2 次元で描画する．viewer は，VASP の入出力で採用されている POSCAR 形式のファイルを読み込み，原子配列を SVG で出力する．なお，ソフト開発は作業の分業化が容易である MVC モデルで作成していく．

2.1 MVC モデルの概要と利点

MVC モデルは，ソフトウェアにおいて取られている手法を担う”Model”，処理系を制御する”Controller”で分業化しやすく，相互にこの利点をもとに，原子

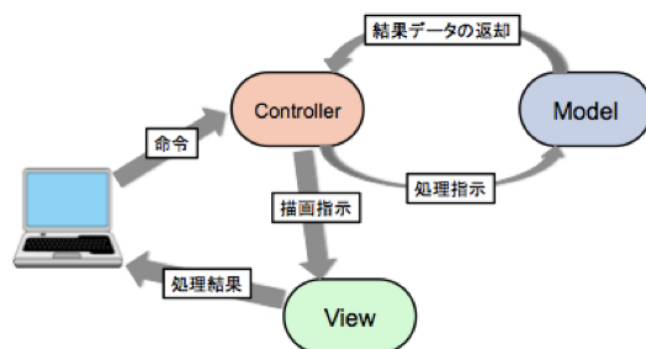


図 4 MVC モデルによる viewer の位置．

2.2 SVG 表示の特徴

SVG は、XML を基盤とする 2 次元画像記述言語であり、イラストレーターで扱うベクターデータである [4]。SVG で表示することで、以下のような利点がある。

- ベクタ形式で画像を表示するため、曲線や文字の拡大・縮小しても画質が劣化せずに、解像度に応じた出力結果を得ることが出来る
- スタイルシートを切り替えることで、特殊な環境下においても目的に応じたグラフィック描画を出力することが可能である
- テキストデータであるため、テキストエディタや xml プロセッサを介したファイルの編集が可能である

なお、原子配列の SVG 表示は、2 次元画像描画ライブラリ”rcairo”を用いて作成していく。

2.3 rcairo を使用する利点

rcairo は、Ruby 言語でベクタ形式の 2 次元描画を容易に実現できるライブラリのことである。このライブラリは、描画コンテキストを用いた API であるため、描画するためのコードを短く簡潔に作成することが出来る。また、複数の出力をサポートすることができるため、出力先のフォーマットに影響されずに描画処理をおこなうことが可能である [5]。

2.4 一般的な原子モデルソフトとの比較

結晶描画に特化したソフトとして、“Medea”や“VESTA”がある。結晶構造や電子・核密度等のデータを読み込んで結晶の形を三次元で可視化できるプログラムである [6]。これらは原子の配置を確認するために、手作業で結晶構造の視点を自由に変えながら、構造全体を視覚的に理解することができる。図には VESTA の画面から切り出した原子配置モデルの一例を示している。3 次元に投影することによって得られた図形である。VESTA の使用には以下の難点があった。

- 三次元で原子配列を表示するため、各層の原子位置が把握しづらい
- 色分けするための層を手動でおこなうため、指定した範囲の正確さに欠け、作業に手間がかかる

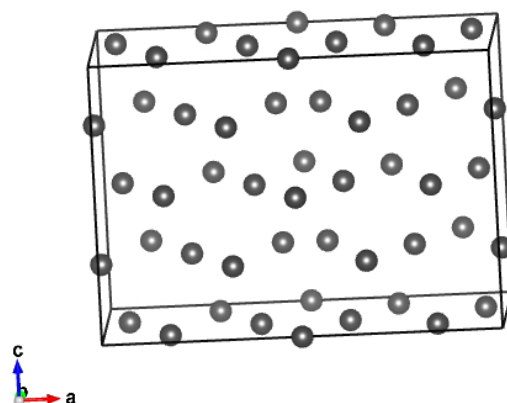


図 5 VESTA で描画した POSCAR_2223 の原子配置 .

これまでの原子配列の構造は，結晶構造描画ソフト VESTA を使用して確認してきたが，三面図を使用して表示することにより，図のように，各面から原子の配置を正確で簡易的に確認できるようになる．

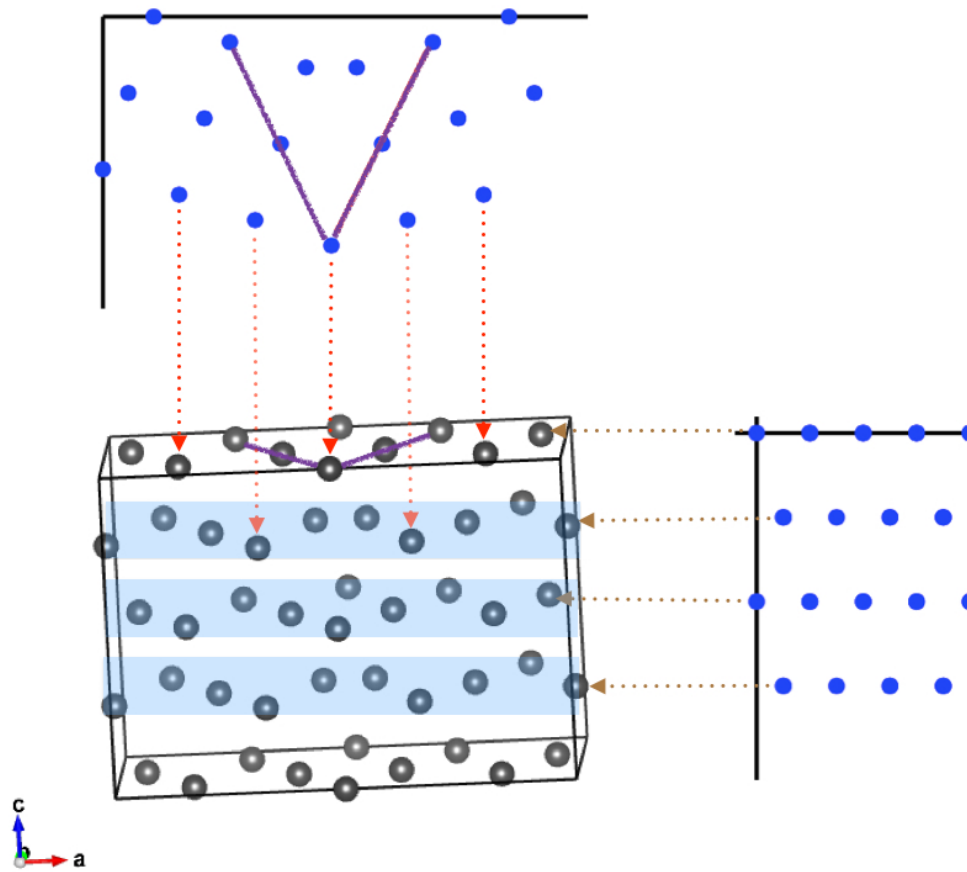


図 6 vesta の投影図を 2 次元化した図

3 ソフトの構成と描画

3.1 外部データの読み込み部

3.1.1 read_pos

```
def read_pos(lines, init_line=8)
  lattice, atom, poscar = [], [], []
  lines[2..4].each{|line| lattice << line.scanf("%f %f %f\n") }

  lines[init_line..lines.length+1].each{|line| atom << line.scanf("%f %f %f\n") }

  atom.each{|i_atom|
    pos=[0.0,0.0,0.0,0.0]
    i_atom.each_with_index{|atom_j,j|
      lx,ly,lz=lattice[j]
      pos[0] += atom_j*lx
      pos[1] += atom_j*ly
      pos[2] += atom_j*lz
    }
    poscar << pos
  }
  return poscar
end
```

外部データである POSCAR.2223 を ARGV で読み込み，lines として定義している．lines の 2 行目から 4 行目の値を読み込んで，配列 lattice に格納している．lines の 8 行目から最後の行の値を読み込んで，配列 atoms に格納している．配列 lattice 内に格納された各々の座標を lx,ly,lz と定義し，格子の座標と原子の座標を掛け算して粒界配列の座標を出し，その値を poscar に格納している．
poscar に格納された原子座標が以下の実行結果である．

```
[Narita-no-MacBook-Air:~/boundary/narita/view] nari% ruby final.rb POSCAR_2223
[5.75101295815, 0.0, 0.0, 0.0]
```

```

[7.668017277916733, 0.6390014395849836, 2.020699977875, 0.0]
[3.834008638383265, 0.6390014395849836, 2.020699977875, 0.0]
[7.029015837611088, 2.556005758968566, 0.0, 0.0]
[4.473010078688911, 2.556005758968566, 0.0, 0.0]
[5.75101295815, 0.0, 4.04139995575, 0.0]
[7.668017277916733, 0.6390014395849836, 6.062099933625, 0.0]
[3.834008638383265, 0.6390014395849836, 6.062099933625, 0.0]
[7.029015837611088, 2.556005758968566, 4.04139995575, 0.0]
[4.473010078688911, 2.556005758968566, 4.04139995575, 0.0]
[6.390014398455645, 4.473010078352149, 2.020699977875, 0.0]
[5.112011517844354, 4.473010078352149, 2.020699977875, 0.0]
[6.390014398455645, 4.473010078352149, 6.062099933625, 0.0]
[5.112011517844354, 4.473010078352149, 6.062099933625, 0.0]
...
[1.2780028794610885, 5.751012958150748, 6.062099933625, 0.0]

```

3.2 原子座標の計算部

3.2.1 identical_atoms

```

def identical_atom(i_atom,j_atom)
  dist=0.0
  3.times{|i| dist += (i_atom[i]-j_atom[i])**2 }
  return true if Math.sqrt(dist)<0.5
  return false
end

```

関数 `identical_atoms` は、2つの POSCAR ファイルに含まれる各々の原子座標の距離を求めて、ある値の大小比較をおこなって真偽を判定する機能を果たしている。具体的には、ファイル内のある原子 `i_atom` ともう片方のファイル内にある原子 `j_atom` の距離が 0.5 より小さい値であれば `true` と判定して返す。そうでなければ、`false` を返して終了する。

3.2.2 mk_deleted_atom

```
def mk_deleted_atom
  mark=[]
  j_max = $pos_after.length
  $pos_before.each_with_index{|i_atom,i|
    update_num=0
    $pos_after.each_with_index{|j_atom,j|
      break if identical_atom(i_atom,j_atom)
      update_num = j
    }
    mark << $pos_before[i] if update_num==(j_max-1)
  }
  return mark
end
```

関数 `mk_deleted_atom` は、原子削除をおこなう前後の POSCAR ファイルを比較して削除された原子のみを別の配列に格納する機能を果たしている。関数内では、始めに更新するための値 `update_num` を 0 と定め、2 つの POSCAR ファイルを `each` 文で繰り返しながら、`identical_atom` で `true` として返されたものを配列 `mark` の方へ格納している。
{{br}} 原子座標ファイル POSCAR_2223 において、配列 `mark` に格納された原子座標が以下の実行結果である。

```
[Narita-no-MacBook-Air:~/boundary/narita/view] nari% ruby final.rb POSCAR_2223
[[6.390014398455645, 4.473010078352149, 2.020699977875, 0.0]
[5.112011517844354, 4.473010078352149, 6.062099933625, 0.0]
[10.863024475994353, 3.8340086387671652, 0.0, 0.0]
[0.6390014403056455, 3.8340086387671652, 0.0, 0.0]
[10.863024475994353, 3.8340086387671652, 4.04139995575, 0.0]]
```

3.3 粒界原子の描画部

3.3.1 draw_backcolor, draw_exes

```
def draw_backcolor
  $context.set_source_rgb(0.8, 0.8, 0.8)
  $context.rectangle(0, 0, $width, $height)
  $context.fill
end

def draw_exes
  $context.set_source_rgb(0, 0, 0)
  [[0,1],[1,0]].each{|line|
    x,y=line[0],line[1]
    [[0,0],[$cx,0],[0,$cy]].each{|c_x,c_y|
      $context.move_to($mv+c_x,$mv+c_y)
      $context.line_to($mv+c_x+x*$scale,$mv+c_y+y*$scale)
      $context.stroke
    }
  }
end
```

関数 draw_backcolor は、原子配列の背景を描画しており、SVG で表示する画面のサイズを明確にするために作成している。また、関数 draw_exes は、原子配列における縦軸と横軸を描画しており、each 文で 3 つ平面に必要な軸をまとめて出力できるようにしている。

3.3.2 open_circle

3.3.3 draw_each_plane, pos.y

```
def draw_each_plane(ind_1,ind_2,c_x,c_y)
  rr = 2
  layer_num = 3
  sel = (ind_1==0 and ind_2==1)? 1 : 0
```

```

[[ $deleted_atoms, [1,0,0], rr*1.3 ], [ $pos_after, [0,0,1], rr ] ].each{|atoms_color|
$context.set_source_rgb(atoms_color[1])
radius = atoms_color[2]
atoms_color[0].each{|pos|
  if open_circle[layer_num-1] < pos[2] && pos[2] < open_circle[layer_num]
    $context.circle($mv+c_x+$adjust*pos[ind_1], pos_y(pos, c_y, ind_2, sel), radius)
    $context.stroke
    $context.set_line_width(0.5)
  else
    $context.circle($mv+c_x+$adjust*pos[ind_1], pos_y(pos, c_y, ind_2, sel), radius)
    $context.fill
  end
}
}

if $pos_before.size==$pos_after.size
$context.set_source_rgb(0, 0.8, 0)
(0..$pos_before.length-1).each{|i|
  $context.move_to($mv+c_x+$adjust*$pos_before[i][ind_1], pos_y($pos_before[i], c_y, ind_2, sel))
  $context.line_to($mv+c_x+$adjust*$pos_after[i][ind_1], pos_y($pos_after[i], c_y, ind_2, sel))
  $context.stroke
}
end

end

def pos_y(pos, c_y, index, select)
  dy = select == 0 ? pos[index] : $pos_max[index]-pos[index]
  return $mv+c_y+$adjust*dy
end

```

関数 `draw_each_plane` で、描画する原子の位置、色、大きさを定めており、関数 `pos_y` は $x - y$ 平面での y 軸を逆転する計算をおこなっている。具体的に、`sel` で描画する平面の判定をおこない、 $x - y$ 平面ならば、関数 `pos_y` で一番大きい y 座標の値から各 y 座標

を差分した値を受け取っている．layer_num は，白抜きする z 層を指定しており，関数 open_circle をもとに白抜きする原子を判定している．また，読み込んだ 2 つの原子配列 ファイル POSCAR のデータサイズが同じならば，各原子の位置に相当する原子との距離を線で表示する．

3.3.4 draw_atoms

```
def draw_atoms
  draw_each_plane(0,1,0,0)
  draw_each_plane(0,2,0,$cy)
  draw_each_plane(1,2,$cx,$cy)
end
```

関数 draw_atoms では，前の関数 draw_each_plane で設定した原子を三面図の規定位置にそれぞれ描画するようにしている．具体的には， $x - y$ 平面を平面図， $x - z$ 平面を正面図， $y - z$ 平面を側面図として三面図を構成させている．

3.3.5 find_max

```
def find_max(pos)
  max = [0,0,0]
  [0,1,2].each{|ind|
    pos.length.times {|i| max[ind] = pos[i][ind] if max[ind] < pos[i][ind] }
  }
  return max
end
```

関数 find_max は，原子の各座標の最大値を探索する機能を果たしている．配列 pos には，関数 read_pos で $(x,y,z)=(0,1,2)$ としてそれぞれの値を読み込んでおり，隣接する値の大小比較をおこなって，各座標の最大値を配列 max に格納している．

3.3.6 main_draw

```
def main_draw(file1,file2,  model_scale = 10)
  lines1 = File.readlines(file1)
  lines2 = File.readlines(file2)
  $pos_before = read_pos(lines1,8)
```



```

$pos_after = read_pos(lines2,8)
$deleted_atoms = mk_deleted_atom

$pos_max=find_max($pos_before)
$pos_max[0].ceil*10
$width,$height = 300,200
$cx,$cy = $width/2.0,$height/2.0
$mv = 10
$scale = 1000
$adjust = $scale/($pos_max[0].ceil*model_scale)
surface = Cairo::SVGSurface.new('view.svg', $width, $height)
$context = Cairo::Context.new(surface)
$context.set_line_width($line_width)

draw_backcolor
draw_axes
draw_atoms
surface.finish
end

```

関数 `main_draw` は、読み込んだ外部ファイルを `lines` として格納し、関数 `read_pos` でつづいた原子座標 `poscar` をファイルごとに分類している。また、`rcairo` で描画処理の基本となるサーフェスとコンテキストを作成し、出力場所及び出力先の形式を定めている。

3.4 三面図による描画

三面図は、立体を正面図、平面図、側面図の三方向からみて投影した図を展開したもので、立体の形状を2次元上で適切に表示することが出来る。三方向から描く各図の位置は厳密に決められており、正面図を物体の最も代表的な面とし、正面図の真上に平面図、正面図の真横に側面図を描く [7]。実際に、三面図を使用して表示した POSCAR_2223 の成功例と失敗例は図のようになる。

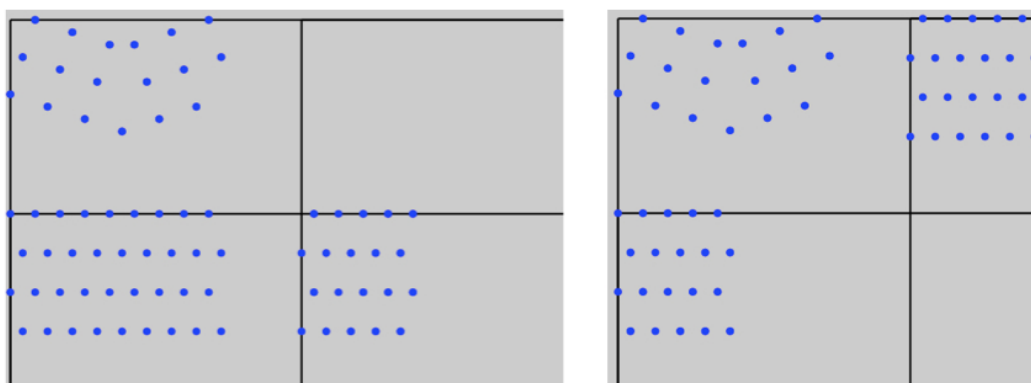


図 7 POSCAR_2223 を表示した三面図の成功例 (左) と失敗例 (右).

4 結果と考察

4.1 各用途に合わせた

4.1.1 構造緩和による

構造緩和は、最安定の原子を動作させる手法である。

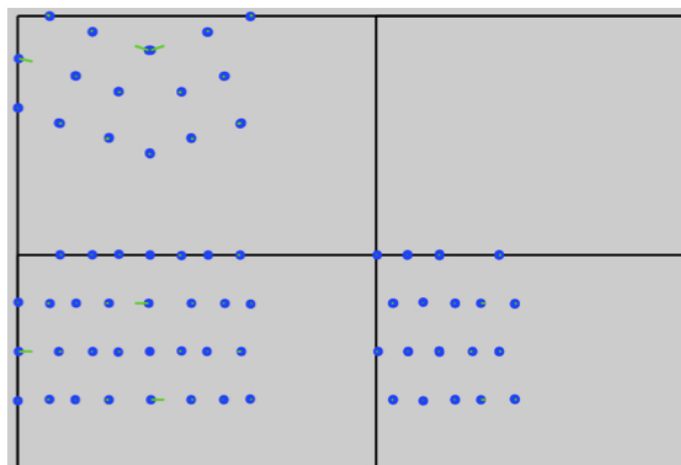


図 8

図中の緑線は、構造緩和をおこなう前後で各原子が移動した経路である。この表示結果により、第一原理計算ソフト VASP による系全体のエネルギーを計算する際の構造緩和に過ちが生じていないかを容易に判断できるようになった。

4.1.2 削除された原子の識別表示。

原子の削除操作は、岩佐の研究で最安定の原子配列の構造を探索するために取り入れた手法であり、削除された原子の位置を視覚的に把握しやすくするためにおこなった図 10 は、削除されたか否かで色分けした原子配置の三面図である。

図中の赤い球が削除された原子に相当する。三面図で描画したことにより、削除された

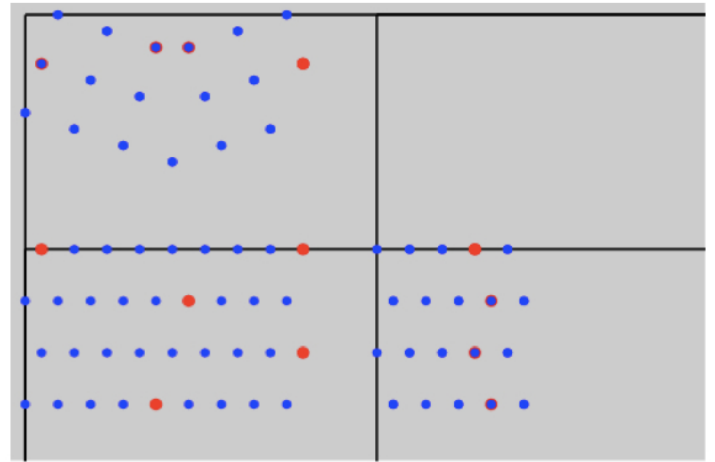


図 9

原子数，並びに各々の配置をすぐに把握することができた．

4.1.3 指定した層の白抜き表示

POSCAR_2223 は 4 層の原子配列で構成されているため，原子配列を上から見た図，すなわち平面図では，原子同士が重なって配置してしまう．したがって，指定した層の原子が上から見てどこに位置するのかを視覚的に確認できるように原子の白抜き処理をおこなった． z 軸の 3 層目を白抜きしたものは，図 12 のように表示された．

4.2 原子構造の改善点

viewer によって三方向の視点で原子配列を表示した結果，構造緩和をおこなうための POSCAR ファイルに原子が一つ不足していることが分かった．不足していた原子の位置は，図 13 の赤枠部分である．

これまでの原子配列は，結晶構造描画ソフト”VESTA”による三次元表示であったため，原子の細かい位置が確認できず，原子が不足していることを認識できなかった．この結果により，構造緩和をおこなう際に使用した POSCAR ファイルに過ちがあったことを発見できた．

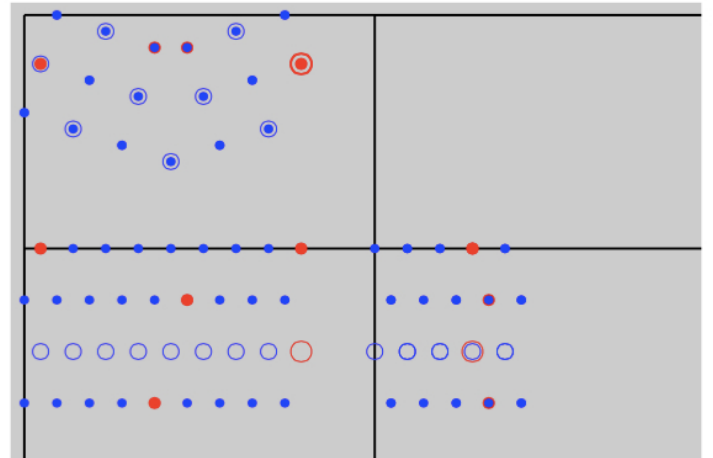


图 10

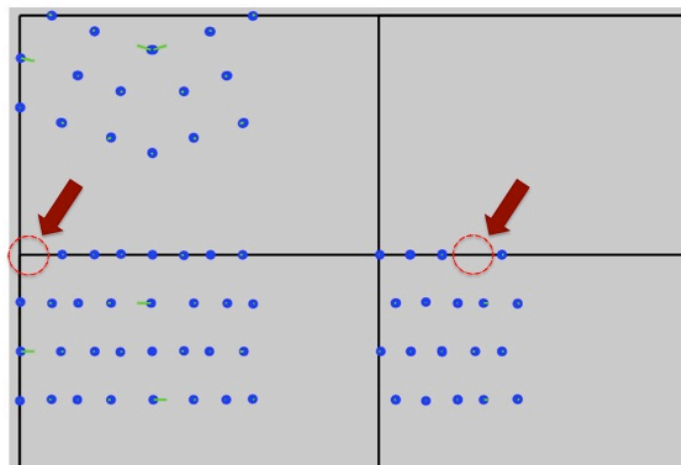


图 11

4.3 考察と今後の課題

5 総括

粒界原子配列を簡易的に視覚化するソフト”viewer”を作成することで，原子配列を 2 次元で，且つ 3 方向からの視点で SVG 表示した．その結果，構造緩和をおこなうための POSCAR ファイルの誤りを発見することが出来た．

6 参考文献

[1] 小傾角粒界粒子シミュレーションの原子ポテンシャル依存性, 八幡裕也 (関西学院大学 理工学部研究科情報科学専攻 修士論文 2015)

[2] 原子削除操作を加えた対称傾角粒界のエネルギー計算, 岩佐恭佑 (関西学院大学 理工学部研究科情報科学 学士論文 2016)

[3] MVC(Model-View-Controller) を理解する, CakePHP, <https://book.cakephp.org/2.0/ja/cakephp>

[4] SVG 入門, 新山祐介, コンピュータサイエンス入門 by 新山祐介, <https://euske.github.io/euske/>

[5] cairo:2 次元画像描画ライブラリ, 須藤功平, Rubyist Magazine-るびま Vol.54 (2016-08), <http://magazine.rubyist.net/?0019-cairo>

[6] VESTA, Koichi Momma(2004-2017), JP-Minerals, <http://jp-minerals.org/vesta/jp/>

[7] 三面図 (機械設計のための基礎製図), 独立行政法人 海上技術安全研究所, NMRI, <https://www.nmri.go.jp/eng/khirata/mechdesign/ch04/ch04.html>