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

小傾角粒界エネルギーにおいて，原子間ポテンシャルを用いたシミュレーションの結果と大槻による実験結果に矛盾が存在する．両者の具体的な相違点は，図に示した Al の (001) 方位の粒界エネルギーの角度依存性から読み取ることができる．Tschopp と Mcdowell によるシミュレーション結果では，0 度及び 90 度付近における対称傾角粒界エネルギーの立ち上がりがそれぞれ異なる傾きになっている [”Asymmetric tilt grain boundary structure and energy in copper and aluminium”, M. A. Tschopp and D. L. Mcdowell, Phil. Mag., Vol. 87 (2007), 38713892.]．その一方，大槻の実験結果では，0 度及び 90 度付近の対称傾角粒界エネルギーの傾きが左右対称になった [A. Otuki, J. Material Science, 40(2005), 3219.].

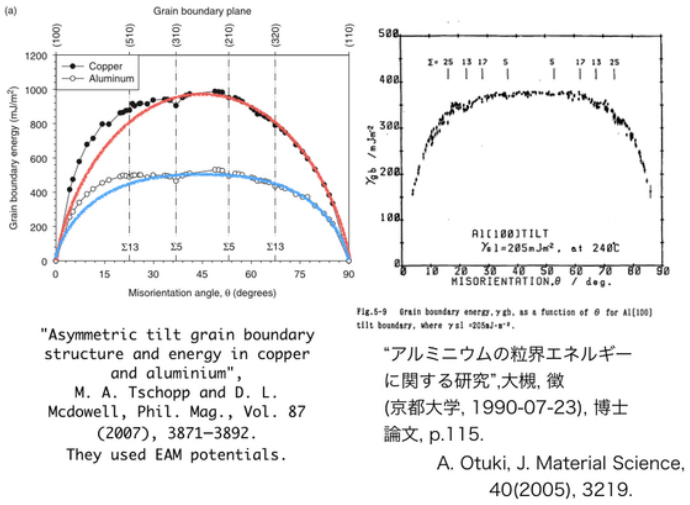


図 1　Al(001) 対称傾角粒界エネルギーのシミュレーションと実験結果の比較.

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 の式は以下のようになる．

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

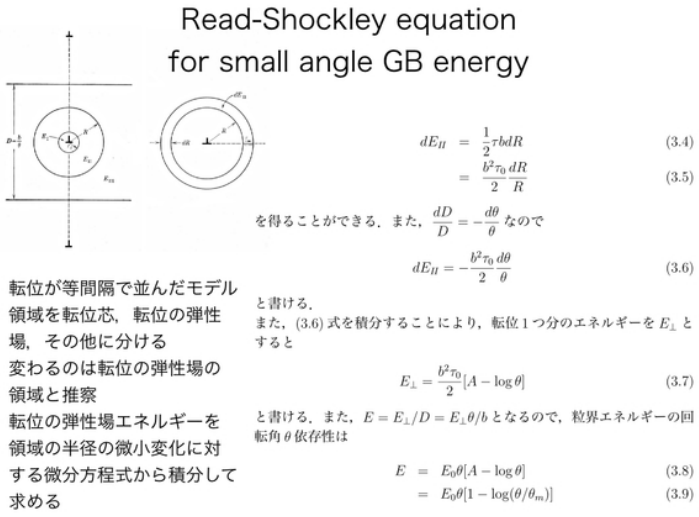


図 2　Read-Shockley の理論予測.

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 [1]．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 [2]．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，図 3 のように原子が全体的に傾いた構造になり，粒界がより低い角度になった状態を計算していた [3].

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．原子配置の確認は Medea や VESTA などの汎用モデル構築ソフトで行われていた．しかし，結果の視覚的な確認には手間がかかるため，日頃のルーチンとしては組み込まれていなかった．この失敗の原因を踏まえて，粒界原子配列を容易に視覚化できるソフトを開発することが本研究の目標である．特に，本研究では小傾角粒界の原子配置の視覚化に特化したソフトを開発する.

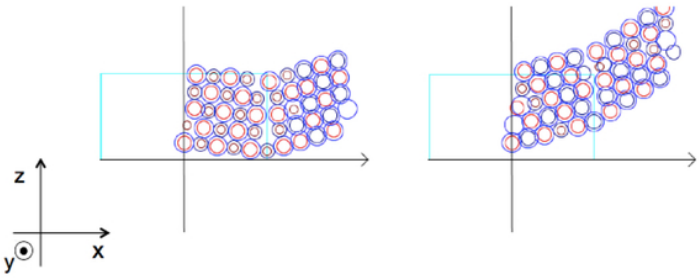


図３　岩佐による誤った構造緩和の結果

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んで，原子配列を SVG 形式で出力する．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．それぞれについて，採用したツールや検討した内容について詳述する．

2.1　一般的な原子モデルソフトとの比較

結晶描画が可能なソフトとして，”Medea”および”VESTA”がある．これらは結晶構造や電子・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 [4]．手作業で結晶構造の視点を自由に変えることにより．原子配置，並びに構造全体を視覚的に把握することができる．図 4 は結晶構造描画ソフト”VESTA”の画面から切り出した原子配置モデルの一例であり，3 次元に投影することによって得られた図形である．これまでの原子配列の構造は，VESTA を用いて確認してきたが，このソフトを使用するにあたって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

そこで，本研究では，原子配列を多面的に描画した三面図で表示するためのソフト開発を試みた．三面図を使用して表示することにより，図 5 のように，各面から原子の配置を直感的かつ簡易に確認できるようになる．

2.2　MVC モデルの概要と利点

本研究で開発するソフトは，作業の分業化が容易である MVC モデルで作成していく．MVC モデルは，ソフトウェアの設計モデルの一つであり，アプリケーションの開発において取られている手法である．MVC モデルは三要素で構成されており，データの処理を担う”Model”，処理結果を画面に表示する”View”，入力情報の受け取って処理機能を制御する”Controller”で設計されている．各々の機能が直交化されているため，開発作業を分業化しやすく，相互の仕様変更による影響を受けずに開発を進めることが出来る [5].

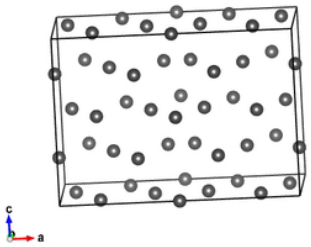


図 4　VESTA で描画した POSCAR_2223 の原子配置.

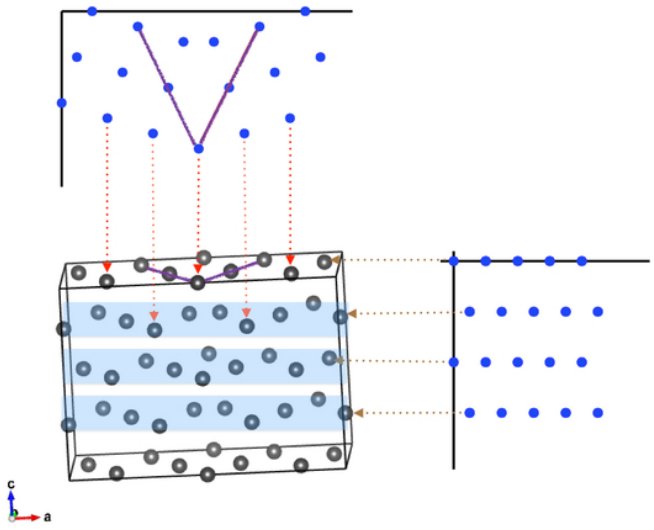


図 5　vesta の投影図を 2 次元化した図

したがって，原子配列の結果を画面表示する機能構築に特化した開発作業が取り組める．

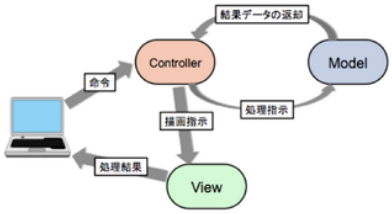


図 6　MVS モデルによる viewer の位置．

2.3　SVG 形式での表示

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 [6]．SVG 形式で表示することにより，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．

2.4 rcairo を使用した描画

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 [7].

3　ソフトの構成

本研究で開発したソフトは，多数の関数で構成されている．ここからは，各関数を機能ごとに詳述する．

3.1　外部データの読み込み部

3.1.1　read_pos

```
1  def read_pos(lines, init_line=8)
2    lattice, atom, poscar = [],[],[]
3    lines[2..4].each{|line| lattice << line.scanf("%f␣%f␣%f\n")  }
4    lines[init_line..lines.length+1].each{|line| atom << line.scanf("%f␣%f␣%f\n"
         ) }
5
6    atom.each{|i_atom|
7      pos=[0.0,0.0,0.0,0.0]
8      i_atom.each_with_index{|atom_j,j|
9        lx,ly,lz=lattice[j]
10       pos[0] += atom_j*lx
11       pos[1] += atom_j*ly
12       pos[2] += atom_j*lz
13     }
14     poscar << pos
15   }
16   return poscar
17 end
```

関数 read_pos は，外部データを ARGV で読み込んでいるファイル lines を受け取って．配列に格納する機能を果たしている．コードの 3 行目では，lines の 2 行目から 4 行目の値を読み込んで配列 latttice に格納しており，コードの 4 行目では，lines 内の 8 行目から最後の行までの座標を読み込んで配列 atoms に格納している． 6 行目からは，配列 atoms を each 文で繰り返し演算をおこない，その中で配列 lattice 内に格納された各座標を lx，ly，lz と定義している．定義した格子の各座標と原子の各座標との積をとることで原子配列の座標をもとめ，その値を配列 poscar に格納している．

原子座標ファイル POSCAR_2223 において，配列 poscar に格納された原子座標を以下の実行結果で示す．各括弧内の数値は，左から順に x 座標，y 座標，z 座標を表している．

```
1 % ruby viewer.rb POSCAR_2223 POSCAR_2223_4
2
3 [5.75101295815, 0.0, 0.0]
4 [7.668017277916733, 0.6390014395849836, 2.020699977875]
5 [3.834008638383265, 0.6390014395849836, 2.020699977875]
6 [7.029015837611088, 2.556005758968566, 0.0]
```

```
 7 [4.473010078688911, 2.556005758968566, 0.0]
 8 [5.75101295815, 0.0, 4.04139995575]
 9 [7.668017277916733, 0.6390014395849836, 6.062099933625]
10 [3.834008638383265, 0.6390014395849836, 6.062099933625]
11 [7.029015837611088, 2.556005758968566, 4.04139995575]
12 [4.473010078688911, 2.556005758968566, 4.04139995575]
13 [6.390014398455645, 4.473010078352149, 2.020699977875]
14 [5.112011517844354, 4.473010078352149, 2.020699977875]
15 [6.390014398455645, 4.473010078352149, 6.062099933625]
16 [5.112011517844354, 4.473010078352149, 6.062099933625]
17 [9.585021596533265, 1.2780028797985992, 0.0]
18 [1.917004319766734, 1.2780028797985992, 0.0]
19 [8.946020157377822, 3.1950071991821813, 2.020699977875]
20 [2.556005758922177, 3.1950071991821813, 2.020699977875]
21 [0.0, 1.9170043193835826, 2.020699977875]
22 [10.863024475994353, 3.8340086387671652, 0.0]
23 [0.6390014403056455, 3.8340086387671652, 0.0]
24 [9.585021596533265, 1.2780028797985992, 4.04139995575]
25 [1.917004319766734, 1.2780028797985992, 4.04139995575]
26 [8.946020157377822, 3.1950071991821813, 6.062099933625]
27 [2.556005758922177, 3.1950071991821813, 6.062099933625]
28 [0.0, 1.9170043193835826, 6.062099933625]
29 [10.863024475994353, 3.8340086387671652, 4.04139995575]
30 [0.6390014403056455, 3.8340086387671652, 4.04139995575]
31 [8.307018717072177, 5.112011518565764, 0.0]
32 [3.1950071992278226, 5.112011518565764, 0.0]
33 [10.22402303683891, 5.751012958150748, 2.020699977875]
34 [1.2780028794610885, 5.751012958150748, 2.020699977875]
35 [8.307018717072177, 5.112011518565764, 4.04139995575]
36 [3.1950071992278226, 5.112011518565764, 4.04139995575]
37 [10.22402303683891, 5.751012958150748, 6.062099933625]
38 [1.2780028794610885, 5.751012958150748, 6.062099933625]
```

3.2 原子座標の計算部

3.2.1 identical_atoms

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

関数 identical_atoms は，読み込んで計算した 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，ある値を基準に大小比較をおこなう判定をしている．コードの 3 行目で，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 の各座標の値を差分し，その値を 2 乗して加えることで原子間の距離を求めている．その距離の値が，0.5 より小さい値ならば true，そうでなければ false と判定して返している．

3.2.2 mk_deleted_atom

```
1  def mk_deleted_atom
2    mark=[]
3    j_max = $pos_after.length
4    $pos_before.each_with_index{|i_atom,i|
5      update_num=0
6      $pos_after.each_with_index{|j_atom,j|
7        break if identical_atom(i_atom,j_atom)
8        update_num = j
9      }
10     mark << $pos_before[i] if update_num==(j_max-1)
11   }
12   return mark
13 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．具体的に，コードの 5 行目で更新するための値 update_num を 0 と定め，2 つの POSCAR ファイルを演算しながら，原子間の距離を求める関数 identical_atom で true として返されたものを配列 mark へ格納している．POSCAR_2223 において，配列 mark に格納された原子座標を以下の実行結果で示す．

```
1 % ruby viewer.rb POSCAR_2223 POSCAR_2223_4
2 [[6.390014398455645, 4.473010078352149, 2.020699977875, 0.0]
3 [5.112011517844354, 4.473010078352149, 6.062099933625, 0.0]
4 [10.863024475994353, 3.8340086387671652, 0.0, 0.0]
5  [0.6390014403056455, 3.8340086387671652, 0.0, 0.0]
6 [10.863024475994353, 3.8340086387671652, 4.04139995575, 0.0]]
```

3.3 粒界原子の描画部

3.3.1 draw_backcolor, draw_axes

```
1  def draw_backcolor
2    $context.set_source_rgb(0.8, 0.8, 0.8)
3    $context.rectangle(0, 0, $width, $height)
4    $context.fill
5  end
6
7  def draw_axes
8    $context.set_source_rgb(0, 0, 0)
9    [[0,1],[1,0]].each{|line|
10     x,y=line[0],line[1]
11     [[0,0],[$cx,0],[0,$cy]].each{|c_x,c_y|
12       $context.move_to($mv+c_x,$mv+c_y)
13       $context.line_to($mv+c_x+x*$scale,$mv+c_y+y*$scale)
```

```
14        $context.stroke
15      }
16    }
17  end
```

関数 draw_backcolor は，原子配列の背景を描画しており，原子配列を表示する画面のサイズを明確にするために挿入されている．また，関数 draw_axes は，原子配列における縦軸と横軸を描画しており，コードの 11 行目にて 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

```
1  def open_circle
2    z_layer=[]
3    layer_max, layer_min = $pos_max[2], 0.0
4    bound=(layer_max - layer_min )/($denominator-1)
5    num,j = layer_min, 0
6    $diff = 0.15
7    while num <= layer_max do
8      z_layer[j] = num
9      num += bound
10     j += 1
11   end
12   return z_layer
13 end
```

関数 open_circle は，白抜きする層の座標を計算して配列に格納している．コードの 3 行目に記述された layer_max は，関数 find_max で得た原子の z 座標の最大値を取っており，layer_min は z 座標の最小値として初期値 0 を与えている．その直後に挿入されている bound は，各層の倍数値を取っている．この倍数値は，z 座標の最大値と最小値の差をとり，差分した数値を関数 main_draw で読み込んだ分母値 denominator で割ることにより値を得ている．コードの 7 行目では，layer_min から layer_max までの範囲で倍数値 bound を加算させていき，加えていった各数値を配列 z_layer に格納している．また，コード内に挿入されている diff は，関数 draw_each_plane にて白抜きする原子座標の近似値をさす．

3.3.3 pos_y

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

関数 pos_y では，x ー y 平面の y 軸を逆転させる計算をおこない．描画する原子の y 座標を返している．具体的に，関数 draw_each_plane の中の sel で，描画する平面を識別した結果をもらい，x ー y 平面ならば，関数 find_max により得た y 座標の最大値から各 y 座標を差分し，その値を dy へ与えている．x ー y 平面以外の平面ならば，最大値からの差分を取らずにそのままの値を返している．

3.3.4 draw_each_plane

```
1  def draw_each_plane(ind_1,ind_2,c_x,c_y)
2    rr = 2
3    sel = (ind_1==0 and ind_2==1)? 1 : 0
4    [[$deleted_atoms,[1,0,0],rr*1.3],[$pos_after,[0,0,1],rr]].each{|atoms_color|
5      $context.set_source_rgb(atoms_color[1])
6      radius = atoms_color[2]
7      atoms_color[0].each{|pos|
8        if $numerator == 0 then
9          $context.circle($mv+c_x+$adjust*pos[ind_1],pos_y(pos,c_y,ind_2,sel),
               radius)
10         $context.fill
11       else
12         if pos[2] < open_circle[$numerator-1]+$diff and open_circle[$numerator
               -1] - $diff < pos[2] then
13           $context.circle($mv+c_x+$adjust*pos[ind_1],pos_y(pos,c_y,ind_2,sel),
                 radius*1.7)
14           $context.stroke
15           $context.set_line_width(0.5)
16         else
17           $context.circle($mv+c_x+$adjust*pos[ind_1],pos_y(pos,c_y,ind_2,sel),
                 radius)
18           $context.fill
19         end
20       end
21     }
22   }
23
24   if $pos_before.size==$pos_after.size
25     $context.set_source_rgb(0, 0.8, 0)
26     (0..$pos_before.length-1).each{|i|
27       $context.move_to($mv+c_x+$adjust*$pos_before[i][ind_1],pos_y($pos_before
             [i],c_y,ind_2,sel))
28       $context.line_to($mv+c_x+$adjust*$pos_after[i][ind_1],pos_y($pos_after[i
             ],c_y,ind_2,sel))
29       $context.stroke
30     }
31   end
32 end
```

関数 draw_each_plane では，描画する原子の位置，色，大きさを決定し，白抜き処理の判定をおこなっている．コードの 2 行目にある値 rr は，原子の半径値を示している．4 行目

から，atoms_color の配列を組んで削除の有無により原子の色と大きさを変えて描画するようにしている．また，8 行目からは，原子の白抜き処理を判定している．具体的に，分子値 numerator が 0 であるかを判定し，true ならば白抜きなしの原子配列を描画する．分子値 numerator が 0 でない，すなわち指定した層の値が numerator に入力されているのならば，関数 open_circle で各層の値を格納している z_layer の配列番号を指定し，近似値 diff で範囲を求め，指定した層の白抜き処理をおこなう．さらに，24 行目からは，読み込んだ 2 つの原子配列ファイル POSCAR のデータサイズを比較している．同じサイズであるならば，2 つのファイルにおける各原子間の距離を線で表示するようにしている．

3.3.5 draw_atoms

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

関数 draw_atoms では，関数 draw_each_plane で決定された原子配列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，x ー y 平面を平面図として第二象限，x ー z 平面を正面図として第三象限，y ー z 平面を側面図として第四象限に出力させている．

3.3.6 find_max

```
1 def find_max(pos)
2   max = [0,0,0]
3   [0,1,2].each{|ind|
4     pos.length.times {|i| max[ind] = pos[i][ind] if max[ind] < pos[i][ind] }
5   }
6   return max
7 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．関数 read_pos にて，原子の x 座標，y 座標，z 座標をそれぞれ pos[0],pos[1],pos[2] の中に格納しており，隣接する値を比較することで，各座標の最大値を探索し，配列 max に格納している．

3.3.7 main_draw

```
1 def main_draw(file1,file2, layer, model_scale = 10)
2   lines1 = File.readlines(file1)
3   lines2 = File.readlines(file2)
4   if layer != nil then
```

```
5       tmp=layer.split('/')
6       $numerator, $denominator = tmp[0].to_f,tmp[1].to_f
7     else
8       $numerator = 0
9     end
10
11    $pos_before, $pos_after = read_pos(lines1,8), read_pos(lines2,8)
12    $deleted_atoms = mk_deleted_atom
13
14    $pos_max=find_max($pos_before)
15    $pos_max[0].ceil*10
16    $width,$height = 300,200
17    $cx,$cy = $width/2.0,$height/2.0
18    $mv = 10
19    $scale = 1000
20    $adjust = $scale/($pos_max[0].ceil*model_scale)
21    surface = Cairo::SVGSurface.new('view.svg', $width, $height)
22    $context = Cairo::Context.new(surface)
23    $context.set_line_width($line_width)
24    draw_backcolor
25    draw_axes
26    draw_atoms
27    surface.finish
28  end
```

関数 main_draw には，描画機能の基盤が設定されており，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．2，3 行目の lines には，読み込んだ各々の外部ファイルを格納しており，コードの 4 行目では，白抜きする層の値が入力されているかを判定している．この判定が true ならば，入力値を/で分別し，それぞれの数値を分子値 numerator, 分母値 denominator として与えている．また，判定が false ならば，numerator に 0 を与えている．11 行目では，読み込んだ 2 つの原子配列ファイルを関数 read_pos で計算し，作成した原子座標 poscar をファイルごとに分類している．具体的に，初めのファイルの方を pos_before，他方を pos_after としている．width，及び height は，出力するスクリーンの大きさを指定しており，18 行目の mv は，出力全体を移動させるための値である．これは，軸上に存在している原子がスクリーンから出てしまうのを防ぐために挿入されている．その直後に挿入されている scale，adjust は，出力する軸，並びに原子の大きさである．

なお，プログラムの文末には以下のコードを記述している．

```
1  model_scale = 1.0/0.12
2  $line_width = 1
3  main_draw(ARGV[0],ARGV[1], ARGV[2],model_scale)
```

model_scale，line_width は，出力する画面に対して軸や原子の大きさを調整するための数値である．

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 次元上で的確に表示することが出来る．開発当初は，上面，正面，側面の配置をまったく気にせずに図 (a) のように配置していた．しかし，三方向から描く各図の位置は JIS 規格で厳密に定めされており，正面図を物体の最も代表的な面と決められている [8]．正面図の真上に平面図，真横に側面図を描くことで三面図が作成される．そこで，図 (b) のとおり最終版では修正して配置している．

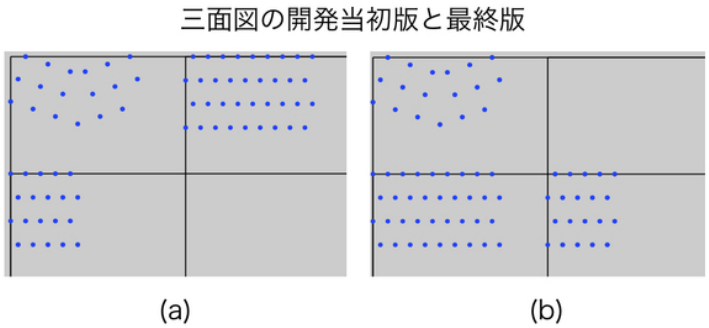


図 7　POSCAR_2223 を表示した三面図の (a) 開発当初版と (b) 最終版.

4　結果と考察

4.1　各用途に合わせた原子配列の表示結果

4.1.1　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る．boundary modeler で 2223 などと指定して作成された原子配置では，真ん中と両端にある粒界近傍で原子位置の極端に近いペアが生成する．このままでは，高いエネルギーとなり緩和がうまく進行しない．そこで，boundary adjuster で原子の削除を行う．削除前後の POSCAR を比較して原子位置を特定する作業を VESTA などでは手動で行うが，この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の表示法を工夫した．図は，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．なお，以下のコードを入力することで safari 上に表示される．

```
% ruby viewer.rb POSCAR_2223 POSCAR_2223_4
% open -a safari view.svg
```

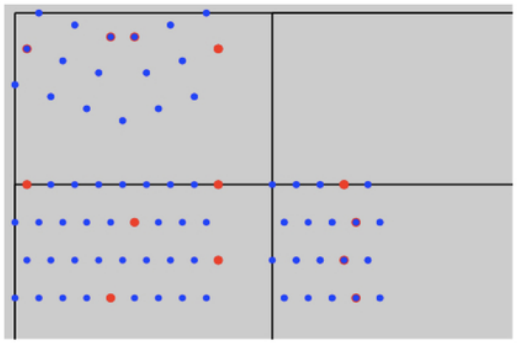


図 8　削除前後の POSCAR の相違をわかりやすくした三面図.

図中の赤い丸が削除された原子に相当する．平面図では青と赤が重なっているが，これは正面図からわかる通り，上下に重なった原子位置で削除の有無が存在するためである．VESTA などの汎用ソフトでは，このような操作は標準で用意されておらず，原子種を変えるなどの工夫によって表示することが必要である．しかし，開発した viewer では二つの POSCAR の原子位置から自動で判定するようにしている．また，三面図で描画したことにより，削除された原子数，並びに各々の配置をすぐに把握することができた．これは後で説明するとおり，粒界構造の確認の時に決定的なエラーに気づかせてくれた．

4.1.2 構造緩和による原子移動の表示

粒界原子配列の構造緩和は，最安定の原子配列を検証するために動かす前後のエネルギーを比較して原子を動作させる手法である．原子の移動を表した図は，同様のソフトに構造緩和前後を示した 2 種類のファイルを入力して表示される．

```
% ruby viewer.rb POSCAR_after POSCAR_before
% open -a safari view.svg
```

以下の図が，構造緩和によって移動した原子配列の三面図である．

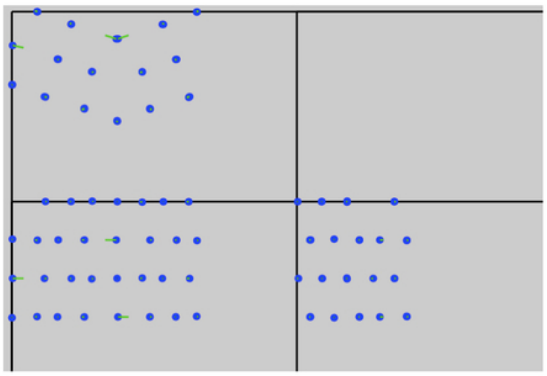


図 9

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．この三面図はSVG で表示されているため，拡大しても解析度が落ちずに鮮明に描画することができ，原子の移動変化をより細かく見ることができる．その結果，原子の正確な移動位置，並びに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．

4.1.3 指定した z 軸の層の白抜き表示

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平面図の中でどこに位置するのかを視覚的に確認するために原子の白抜き処理をおこなった．白抜き処理は，読み込む 2 つのファイル名の後に白抜きする層の段階数を入れることで表示される．層の段階数は上層から順に数字を割り振っており，分母が層の総数，分子が白抜きしたい層である．なお，読み込む 2 つのファイル名の後に数値を入力しない，若しくは分子に 0 の値を入力することにより，白抜きなしの三面図を表示することが可能である．

以下の実行は，4 層で構成されている POSCAR_2223 の第 1 層目を白抜き処理する際に入力したものである．

```
% ruby viewer.rb POSCAR_2223 POSCAR_2223_4 1/4
% open -a safari view.svg
```

このときに表示された原子配列の三面図は．以下の図である．

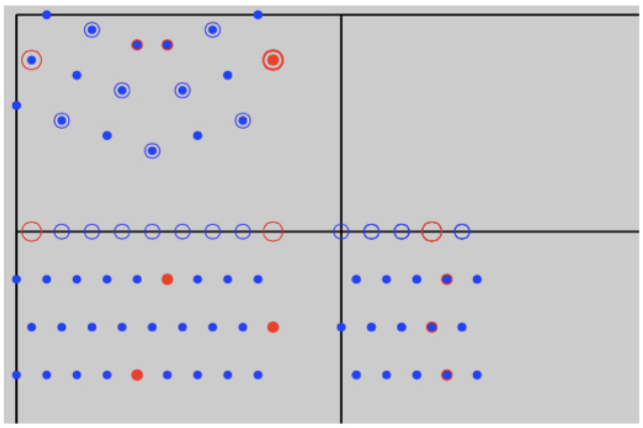
この表示結果により，平面図と正面図の対応した位置が正確に判断できるだけでなく，平面図と側面図で対応した原子の位置を視覚的に判断することが可能になった．

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．

この表示が，viewer による計算，描画の誤りでないかを検証する必要があるため，ファイル POSCAR_after の原子配列を VESTA で三次元化して確認をおこなった．

その結果，図のように各々の位置で対応する原子がある中で，赤枠で括った部分には対応した原子が存在していないのが分かった．したがって，三面図による不足した原子の発見はソフト内の過ちではないことが視覚的に立証できた．



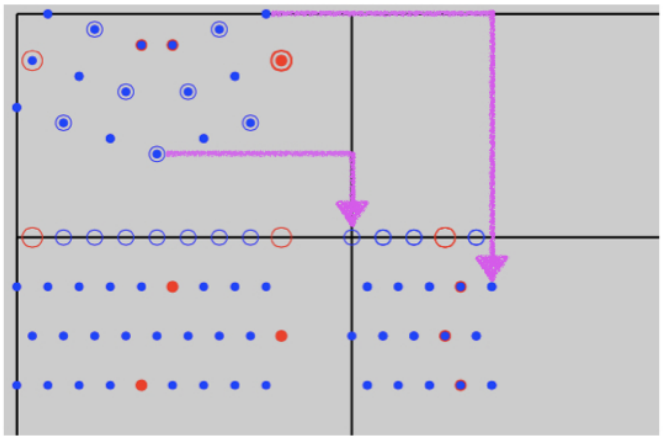
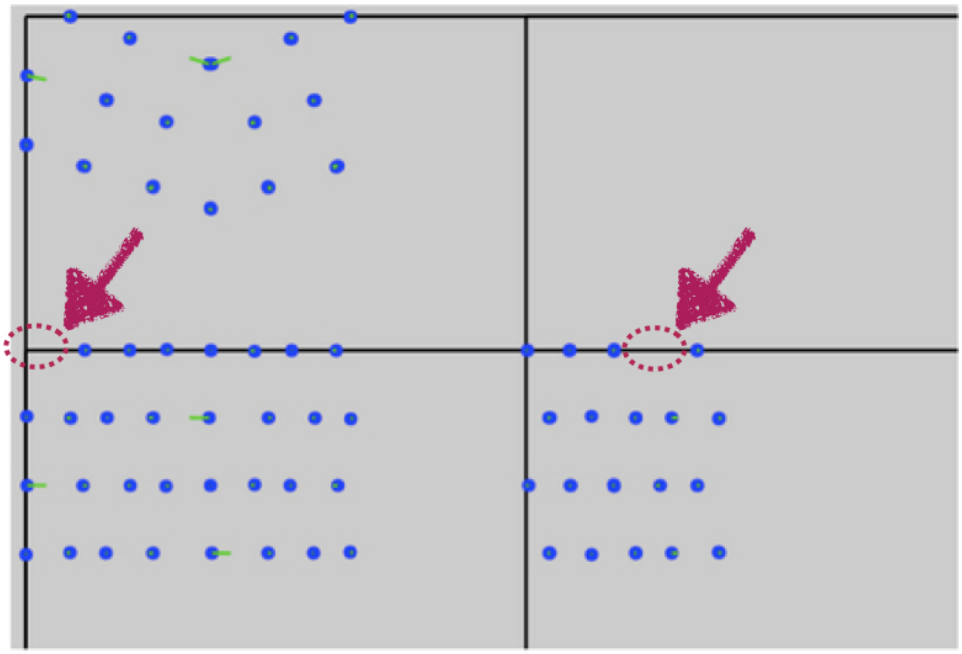


図 11



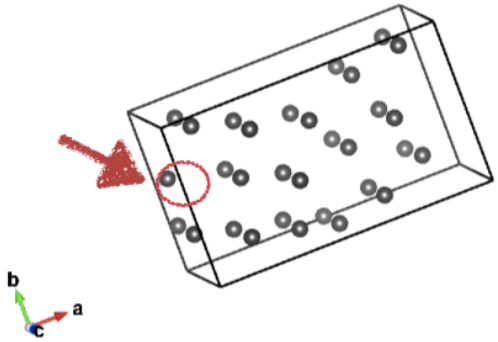


図 13

4.3 考察と今後の課題

三面図を利用して原子配列を表示したことで，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．これは，今までの原子配列が結晶構造描画ソフト”VESTA”による三次元表示であり，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ためである．不足した原子の発見により，粒界原子配列の構造緩和をおこなう計算を見直す必要がある．また，本研究ではPOSCAR_2223 を基準として様々な描画を出力できるソフトを開発したが，より大きな粒界原子配列を視覚的に検証できるように改良していかなければならない．さらに，三面図の配置では，正面図を物体の最も代表的な面と規定されているが，原子配列の表示で最も重要な面は，上から見た図，すなわち平面図であることが開発後に分かった．したがって，今後は三面図の配置構成を以下の図のように変更して視覚的検証をおこなっていく．
{{attach_view(boundary_narita.019.jpeg)}}

5　総括

本研究は，最安定の構造をとるためにおこなう原子の削除操作，及び構造緩和を視覚的に，且つ容易に把握できるソフトを開発することが目的であった．作成したソフトは，rcairo を用いて 2 次元で原子配列を描画し，SVG 形式の三面図で表示するようにした．

開発を進めた結果，様々な用途に合わせて原子配列を表示することが可能となった．まず，削除操作を表した原子配列の表示では，削除の有無により原子の色と大きさを変えて描画したことで，削除された原子の個数，並びに各位置を視覚的に把握することができた．また，構造緩和による原子移動の表示では，構造緩和前後で原子が移動した経路を示すことができ，構造緩和に過ちが生じていないかを容易に確認できるようになった．さらに，指定した z 軸の層の原子を白抜きする機能を追加したことで，上面から見た各層の原子位置を正確に把握できるようになった．三面図による描画，並びに VESTA による視覚的検証の結果として，構造緩和前後の原子座標を格納した POSCAR ファイル内に原子が一つ不足していることが発見できた．

今後は，粒界原子配列の構造緩和をおこなう計算を見直す研究を進めていく必要がある．さらに，本研究では，一つの原子配列ファイルをもとに様々な表示ができるソフトを開発したが，このソフトをより大きな粒界原子配列を表示できる機能へ改良していかなければならない．

6　参考文献

参考文献

[1] VASP による粒界エネルギーの第一原理計算, 村上成那 (関西学院大学 理工学部研究科情報科学専 攻 修士論文 2014)

[2] 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)

[3] 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 学士論文 2016)

[4] VESTA, Koichi Momma(2004-2017), JP-Minerals.

[5] MVC(Model-View-Controller) を理解する, CakePHP.

[6] 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.

[7] 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).

[8] 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.