転位の原子レベルでの視覚化

関西学院大学 理工学部
情報科学科 西谷研究室
7629 釜下翔

2011年2月21日

概 要

結晶中の 1 次元欠陥である転位およびその部分転位に挟まれれた積層欠陥は，模式図で理解するのは容易であるが，現実の原子位置から理解するのは難しい．ところが，原子レベルのシミュレーションを理解するときには，そのモデル作成のために，原子サイトの様子を適切に表示し，さらに多方向から検討を加えることが不可欠である．本研究では，fcc 構造における転位が入った時の様子を 3DCG によって視覚化するシステムの開発と，それを用いた原子レベルでの転位構造の理解を目的とした．

まず，刃状転位，らせん転位を静止画で視覚化した．これにより，対称傾角粒界で等間隔に現れる転位を，原子レベルのシミュレーションから理解するときも，視覚的観点から補うことが可能となり，研究効率が高まった．

次に静止画だけでなくアニメーションの製作も行った．静止画で転位が起こる前後のモデルをただ比較するよりも，アニメーション化により，流動的に転位というものを捉え易くなったと思われる．アニメーションの Maya での表示，保存方法などについてもまとめてあるので，今後の他学生の研究にも役立てていただけるのではないかと思う．

さらに，積層欠陥や部分転位について，様々な視点（方向）からの視覚化を行った．近年は，高分解能顕微鏡像で原子の様子を見ることは可能であるが，その視点は固定されている．今回の研究結果によって，さまざまな視点（方向）からの確認が可能となり，理論面での学習を行う際にはとても有効なものとなった．

目 次

第1章　序論

実際の物理学の現象において理論，数式を考えるには，そのモデル化が重要であるが，平面の模式的なモデルではその理解が困難となる場合がある．

金属などの物質を原子単位で見たときに，原子球が三次元的に規則正しく並んでいると思われているかもしれないが，実際の結晶には何らかの乱れが含まれていることがある．それらを総称して，格子欠陥と呼ぶ．格子欠陥には点欠陥や，線欠陥，面欠陥がある．線欠陥は特に転位と呼ばれ，材料の変形を左右する重要な欠陥である．また fcc 構造中に入った転位は，より幅のせまい部分転位に分解することが知られている．この 2 本の部分転位は広がり，その間に積層欠陥と呼ばれる面欠陥を形成する．本研究では，文献などでは平面的，模式的なモデルでしか表されていないため想像し難いようなモデルを，3DCG（3 Dimension Computer Graphics）モデルで視覚化させることで，より明確なイメージを与え，直感的に理解できるようにする目的で行う．

物理現象を視覚化させるツールとして，3DCG ソフトウェアである Maya を使用した．Maya は映画，ゲーム，CM の制作にも使用されるプロ仕様のハイエンドソフトであり，非常に高度な機能を有している．その一つに MEL（Maya Embedded Language）というスクリプト言語がある．Maya はその GUI（Graphics User Interface）の全てを MEL で実行することが可能となっている．結晶の格子モデルや物理現象を再現するにあたり，その理論，数式に基づいた数値計算を行う必要があるが，MEL は Maya 独自のスクリプト言語であり，MEL を利用する環境や構文が数値計算に向いていない．そこで本研究では，MEL より計算環境の優れた Ruby によって，数値計算を行い，その結果を反映させた MEL スクリプトを自動生成させ，最終的に Maya で MEL スクリプトを読み込むことで，視覚化を実現させた．

結晶中の 1 次元欠陥である転位およびその部分転位に挟まれれた積層欠陥は，模式図で理解するのは容易であるが，現実の原子位置から理解するのは難しい．ところが，対称傾角粒界で等間隔に現れる転位や，Mg の長周期構造中に現れる積層欠陥等を，原子レベルのシミュレーションから理解するときには，そのモデル作成のために，原子サイトの様子を適切に表示し，さらに多方向から検討を加えることが不可欠である．本研究では，fcc 構造における転位が入った時の様子を 3DCG によって視覚化するシステムの開発と，それを用いた原子レベルでの転位構造の理解を目的とする．また加えて，積層欠陥において fcc 構造が hcp 構造に変化する事象を，視覚化することによって理解し易いものとすることも目的とする．

第2章　物理学的背景

2.1　転位（dislocation）

金属材料はミクロレベルで見ると三次元的に構成原子が規則正しく並んだ完全結晶であるが，実際の結晶には何らかの規則の乱れが含まれている．それらを総称して格子欠陥と呼ぶ．格子欠陥には 0 次元的な欠陥の点欠陥や，1 次元的な欠陥の線欠陥や 2 次元的な欠陥の面欠陥がある．線欠陥は転位と呼ばれ，材料の変形を左右する重要な欠陥である．代表的な転位として刃状（edge）転位とらせん（screw）転位がある．

刃状転位について述べる．図 2.1 には 1 原子間距離のすべりがすべり面の左半分で起こっているが，右半分では起こっていないような単純立方格子を示す．すべっている部分とすべっていない部分との境界を転位という．その位置は，図 2.2 に示されるように，結晶の上半分に余分に挿入された垂直な半平面をなす原子面（extra half plane）の端で示される．転位の近くでは結晶は大きいひずみを受けている．単純な刃状転位は，すべり方向に垂直に，すべり面上をどこまでも続いている [1,pp.644-46].

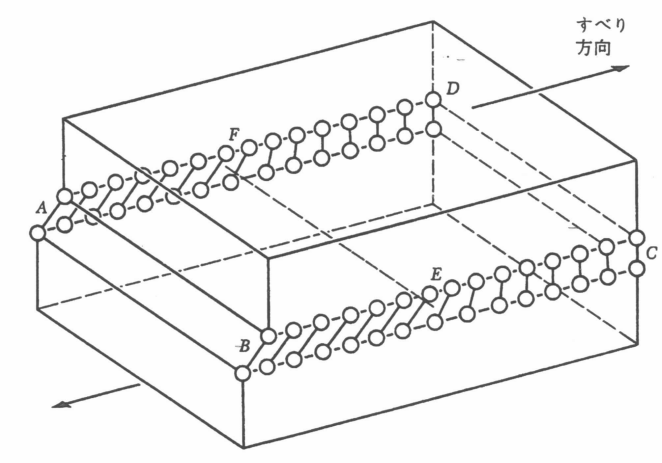


図 2.1: すべり面 ABCD にある刃状転位 EF．原子が格子定数の半分以上変位しているようなすべりの生じている領域 ABEF と，変位が格子定数の半分以下であるすべりの生じていない領域 FECD が示されている [1].

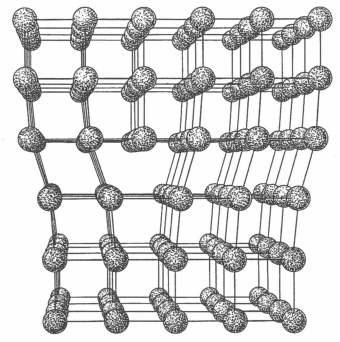


図 2.2: 刃状転位の構造．y 軸の上半分に余分の原子価を挿入したために，変形していると考えられる．この挿入で，上半分の原子は圧縮され，下半分の原子は引き伸ばされる [1].

一方，らせん転位とはすべった部分とすべらない部分の境界がすべり方向に平行な転位をいう．らせん転位は結晶をナイフで途中まで切り込んで，切り口に平行に一方を１原子間距離だけずらせることによってつくられたと考えることができる．らせん転位があると，次々に並んだ原子面は変形してらせんの面の形となる．[1,pp.647-48].

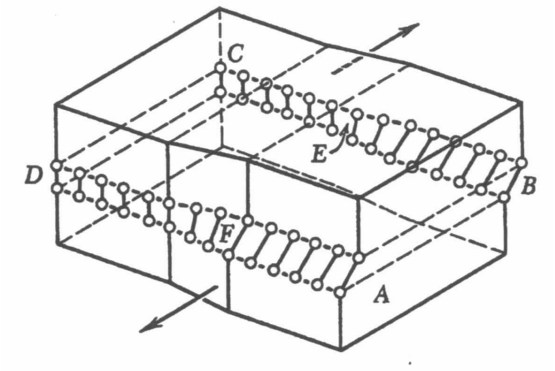


図 2.3: らせん転位．すべり面の一部分 ABEF が転位線 FE に平行にすべっている [1].

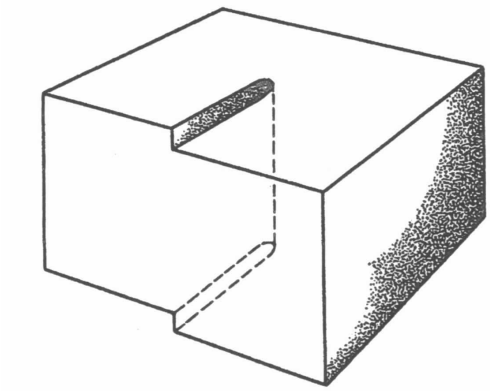


図 2.4: らせん転位の別の表し方．転位を示す縦の破線がひずみを生じた物質で囲まれている [1].

2.2　バーガース・ベクトル（Burgers vector）

転位というのはすべりを起こした領域の境界であり，または空格子点などの板状に凝縮した領域の境界であることを述べた．すべりの場合にはすべり面の一方の物質を他方に対してすべり面に平行に変位させる．これに対し凝縮の場合には凝縮面にほぼ垂直にさせている．この両者を組み合わせると，結晶内に任意の面を考えて，片側の結晶を他方に対して格子ベクトル b だけ変位させると考えることによっても転位を定義できる．

実際，図 2.5〔a〕,〔b〕,〔c〕,〔d〕に示すように，結晶内に任意の方向に切れ目を入れて，一方の結晶を b だけ変位させると，原子がちょうど重なるところとすき間とができる．すき間のところは原子を取り去ったのと同じすき間になっている．したがって，原子の重なるところには空格子点をもってきて原子を取り去り，すき間には格子間原子を流し込ませると，もとと同じ結晶構造にもどり，切れ目のあとはきれいになくなる．変位のしわ寄せは境界のところにだけ残るのであるから，切れ目の方向がどのように変わっても，切れ目の境界が同じで，変位をあらわす格子ベクトル b が同じならば，境界にしわ寄せされるひずみの模様は同じものとなる．[2,pp.24-5].

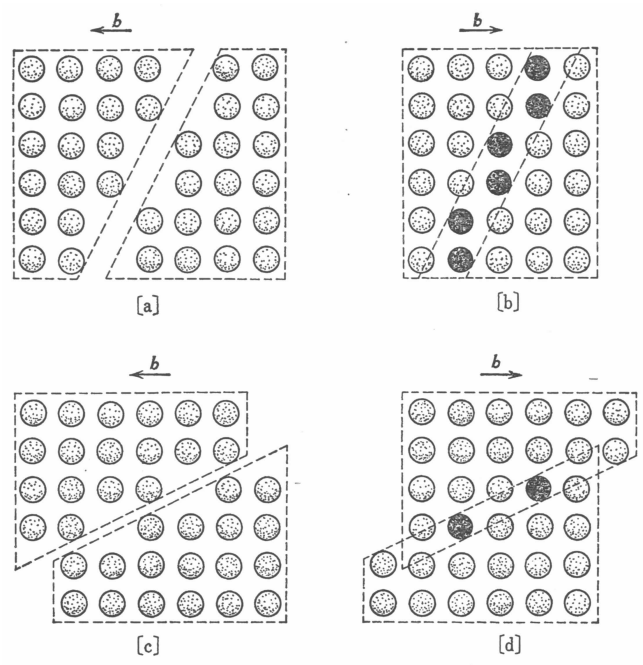


図 2.5: 結晶に任意の方向の切断面を入れ，相対的に b だけ変位させたときの原子配列.〔a〕,〔c〕のすき間には原子を詰めると完全な結晶となる.〔b〕,〔d〕では黒丸の原子は，ちょうど 2 個重なっているので，原子を取り去れば完全結晶になる [2].

この定義で変位をあらわすベクトルbは転位を特徴づける重要な量で，その重要性を最初に指摘したBurgersの名前をつけてバーガース・ベクトルと呼ぶ[2,p.26].

2.3 バーガース・サーキット（Burgers circuit）

実際の結晶のよい領域内の原子を，つぎつぎと結んで，閉じた１つの径路を作る．これをバーガース・サーキットと呼ぶことにする．理想結晶にも対応する径路を考えることができる．この理想結晶中の対応する径路を径路を対応サーキットと名付ける．

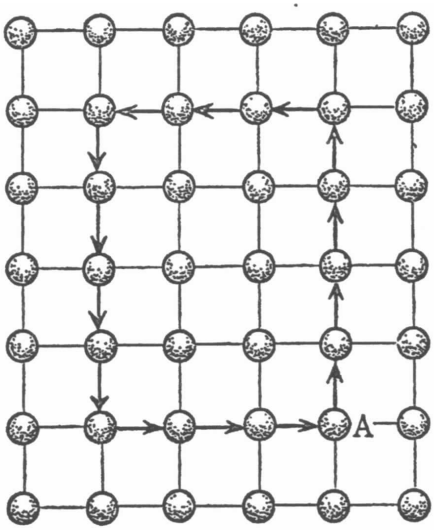


図 2.6: よい結晶内のバーガース・サーキット[2].

いま，図2.6のように，一つのバーガース・サーキットがあって，サーキットをふちとする局面の近くではすべてよい結晶であるとすると，対応サーキットもまた閉じている．しかし，もしバーガース・サーキットが，悪い結晶を含んでいる円筒形の面を取り巻く場合には，バーガース・サーキットの近くはよい結晶であっても，対応サーキットは閉じるとは限らない．そこで転位をつぎのように定義する．対応サーキットが閉じないとき，バーガース・サーキットは転位を囲むという．バーガース・サーキットを時計の針の進む方向と逆にとったときに，対応サーキットを閉じさせるのに必要なベクトル（対応サーキットの終点から始点へのベクトル）をバーガース・ベクトルと呼ぶ．正の刃状転位，らせん転位のまわりのバーガース・サーキット，対応サーキットをそれぞれ図2.7〔a〕,〔b〕，図2.8〔a〕,〔b〕に示す[2,p.29].

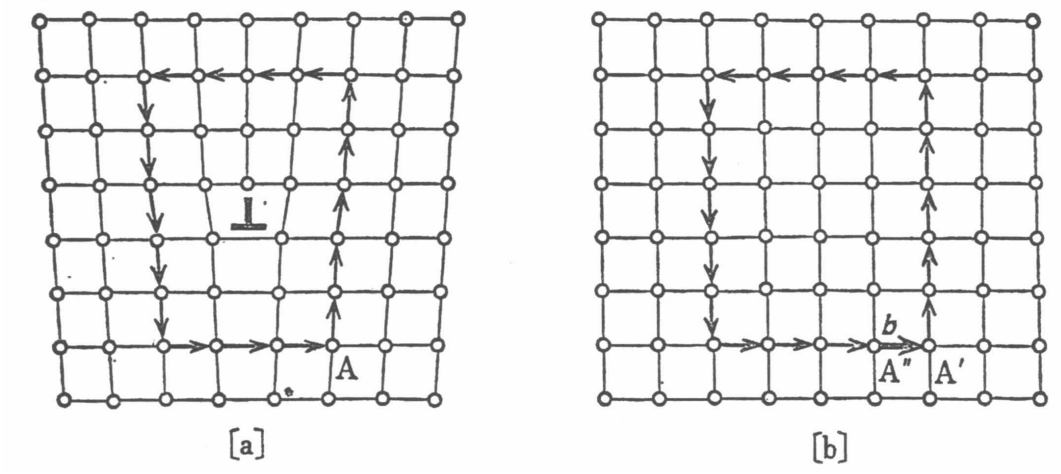


図 2.7: 刃状転位まわりのバーガース・サーキット〔a〕と対応サーキット〔b〕[2].

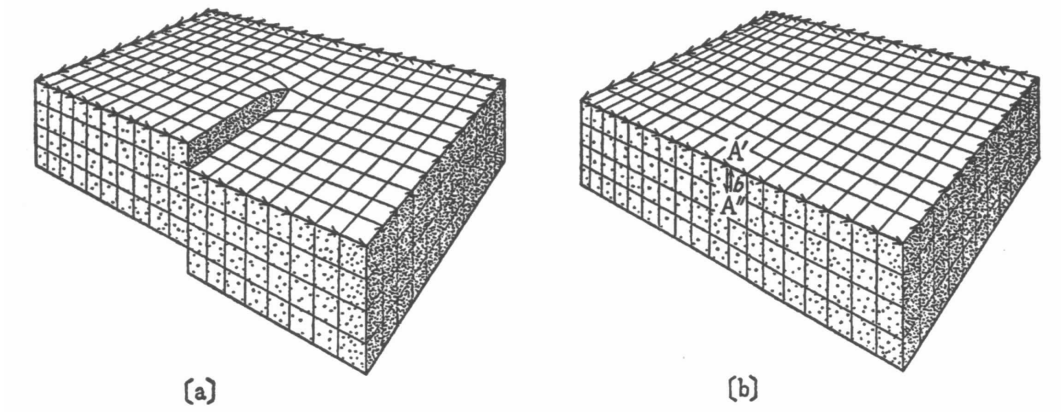


図 2.8: らせん転位まわりのバーガース・サーキット〔a〕と対応サーキット〔b〕[2].

2.4 積層欠陥（stacking fault）と部分転位（pertial dislocation）

面心立方結晶では図 2.9 のように立方体の角と面の中心に原子の並んだものが単位となっている．この立方体が積み重ねられて，原子が周期的に並んだ構造が面心立方結晶（次から fcc; face centered cubic とする.）である．

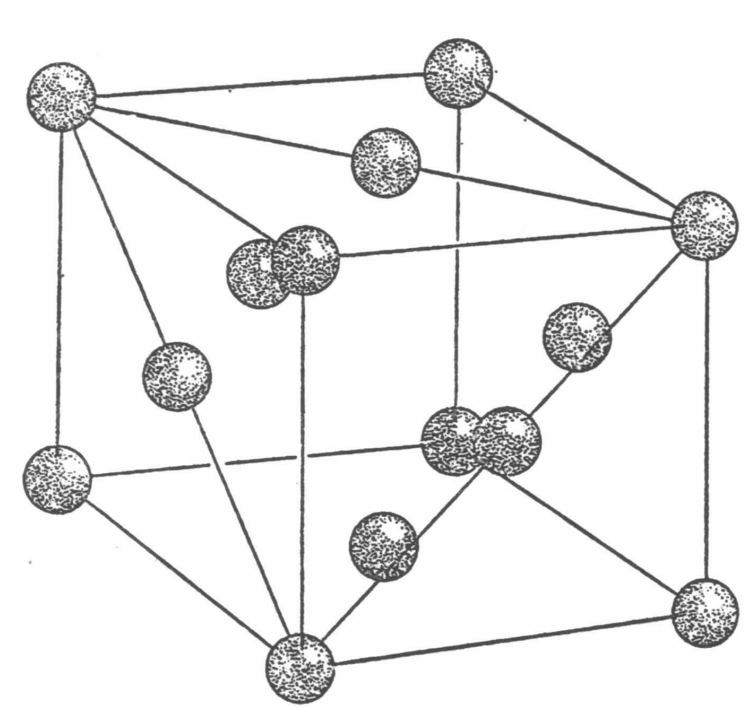


図 2.9: fcc の単位胞 [2].

この結晶で原子がもっとも密に並んでいる方向は図 2.9 の面の対角線の方向である．また原子がもっとも密に並んでいる原子面は，図 2.9 に示すように，どの面上でも対角線で交わる面である．原子がもっとも密に並べでいる面では，図 2.10 に示すように球をもっとも密に並べたものと同じ原子配列になっている [2,p.141].

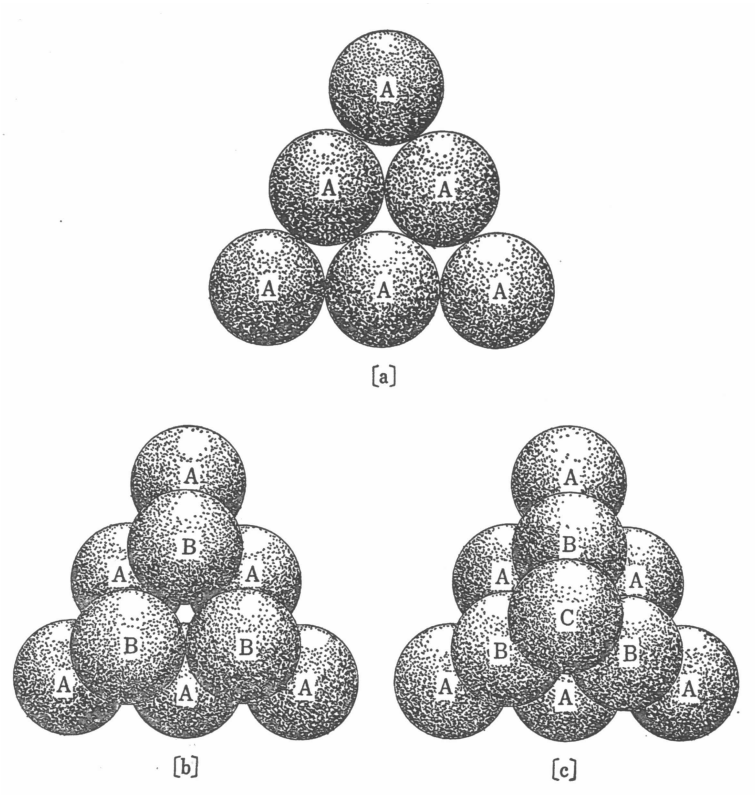


図 2.10: fcc の最密原子面の積み重なり方 [2].

fcc は別な見方をすると，図 2.10 のような原子面が積み重なっているのであるが，その積み重なりかたは，球を積み重ねるのと同じようにくぼんだ位置，すなわち，図 2.10〔b〕のように B の位置に積み重なっている．さらにその上の原子面は C の位置にくる．

しかし，球の場合には，A の原子面のすぐ上に B を重ねても，また，すぐ原子面 C を重ねても同じ安定性をもっている．また，A のすぐ上に B があるとき，そのつぎに C を重ねても，また A をもってきても，同じく安定である．その AB の上にまた A が重なっているような構造を六方最密結晶（次から hcp; hexagonal close packed とする.）という．これに対して，fcc の場合には，A の上に B がきているときには，つぎの原子面は C の位置にくるほうが安定なのである．しかし，Au, Ag，Cu のような代表的な fcc 金属のイオンは，野球のボールのようなもので，重なり合うと反発力を及ぼし合っている．またイオンの間を自由に動きまわっている電子は，イオンをできるだけ小さな体積につめこもうとする力を及ぼしている．したがって，イオンがどの位置に並ぶと安定であるかという問題は，球を密に積み重ねるときの安定性から，ある程度想像することができる．すなわち，AB のつぎに A がきたとしても，C がきたときに比べてエネルギーの増加はわずかであろうと想像できる [2,p.141].

いまこの原子が密に並んだ面にそって，B の位置にある原子面から上の結晶をずらせるとしよう．上の結晶がずべり面に沿って最近接原子対を結ぶベクトル$\overrightarrow{BB}$だけずれると，もとの結晶とまったく同じ原子配列になる．しかし，$\overrightarrow{BB}$にいかず$\overrightarrow{BC}$にずれる場合には，図 2.11 のように鞍部を通っていくことができる．また C の位置まで変位した状態は安定な状態であるから，ここまで変位して止まることもありうるであろう．fcc では最密原子面が，

……A B C A B C A B C A B C A B C……

または

……C B A C B A C B A C B A C B A……

という順に積み重なっているのであるが，B の原子面が C の位置までずれると

……A B C A B C A C A B C A B C A……

と hcp のようになってしまう．この縦線の位置で原子面の積み重ねの順番が狂っている．このような原子面の積み重ねの順番の狂いを積層欠陥と呼ぶ．

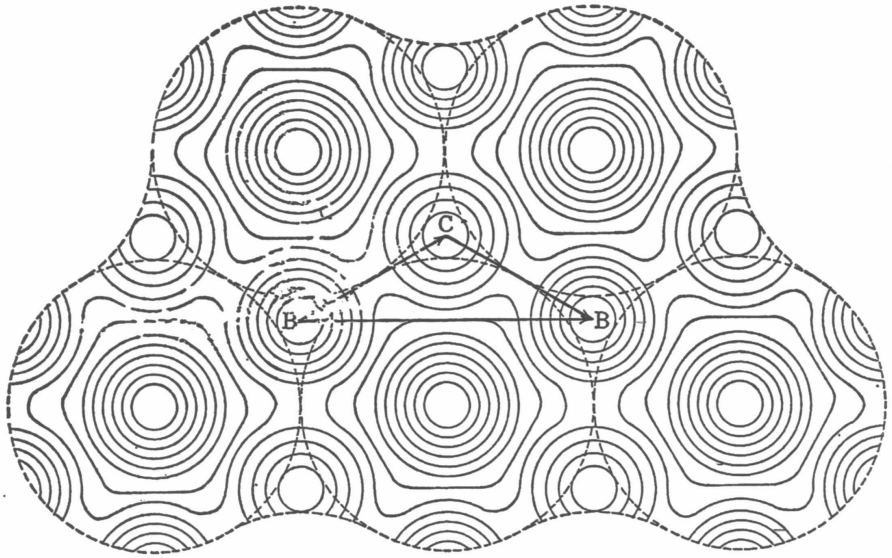


図 2.11: fcc の最密原子面がずれるときのエネルギー変化 [2].

積層欠陥の存在が許される場合には，バーガース・ベクトルが，格子ベクトルと一致しなくてもよい．すなわち fcc の上の例から知られるように，格子ベクトル$\overrightarrow{BB}$よりも小さな$\overrightarrow{BC}$というベクトルだけすべりを起こさせるとき，すべりが部分的にしか起きなければ，すべった領域とまだすべっていない領域の境界にバーガース・ベクトルが$\overrightarrow{BC}$であるような転位が形成される．このように格子ベクトルと異なったバーガース・ベクトルをもつ転位を部分転位（不完全転位）と呼ぶ．[2,pp.141-43]

2.5 fccの部分転位

fccの積層欠陥は最密原子面 $\{111\}$ にそって作られる．いま $(1\bar{1}1)$ 面上の完全転位を考える．

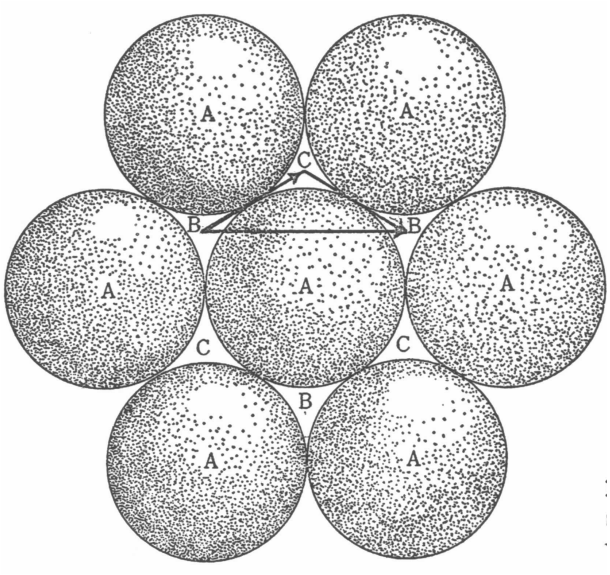


図 2.12: fccのすべりにおける原子移動過程 [2].

完全転位のバーガース・ベクトルは最密原子列に平行で，その方向の原子間隔に等しく，$\frac{a}{2}[110]$ ，$\frac{a}{2}[011]$，$\frac{a}{2}[10\bar{1}]$ のいずれかである．いまある転位のバーガース・ベクトルは $\frac{a}{2}[110]$ とする．この記号はベクトルの x，y，z 成分がそれぞれ $\frac{a}{2}$，$\frac{a}{2}$，0であることを示している．このベクトルは図2.12の$\overrightarrow{BB}$にほかならない．しかし，$\overrightarrow{BC}$までずれると一応安定な位置になるので，$\overrightarrow{BB}$=$\overrightarrow{BC}$+$\overrightarrow{CB}$と変位することが考えられる．上に説明した記号を用いると

$$\frac{a}{2}[110] = \frac{a}{6}[121] + \frac{a}{6}[21\bar{1}] \tag{2.1}$$

に分解することができる．$\frac{a}{6}[121]$ 変位した状態は積層欠陥であるから，積層欠陥の帯をはさんで2本の部分転位に分解したということができる．このとき，すべり面のすぐ下の原子を白丸，すぐ上の原子を黒丸で書くと，図2.13のようになっている．〔a〕は完全転位のとしては刃状転位,〔b〕はらせん転位の場合である．2.1式から知られるように部分転位のバーガース・ベクトルは完全転位のバーガース・ベクトルと平行でない．また一方の部分転位は純粋な刃状転位でもらせん転位でもない [2,pp.155-56].

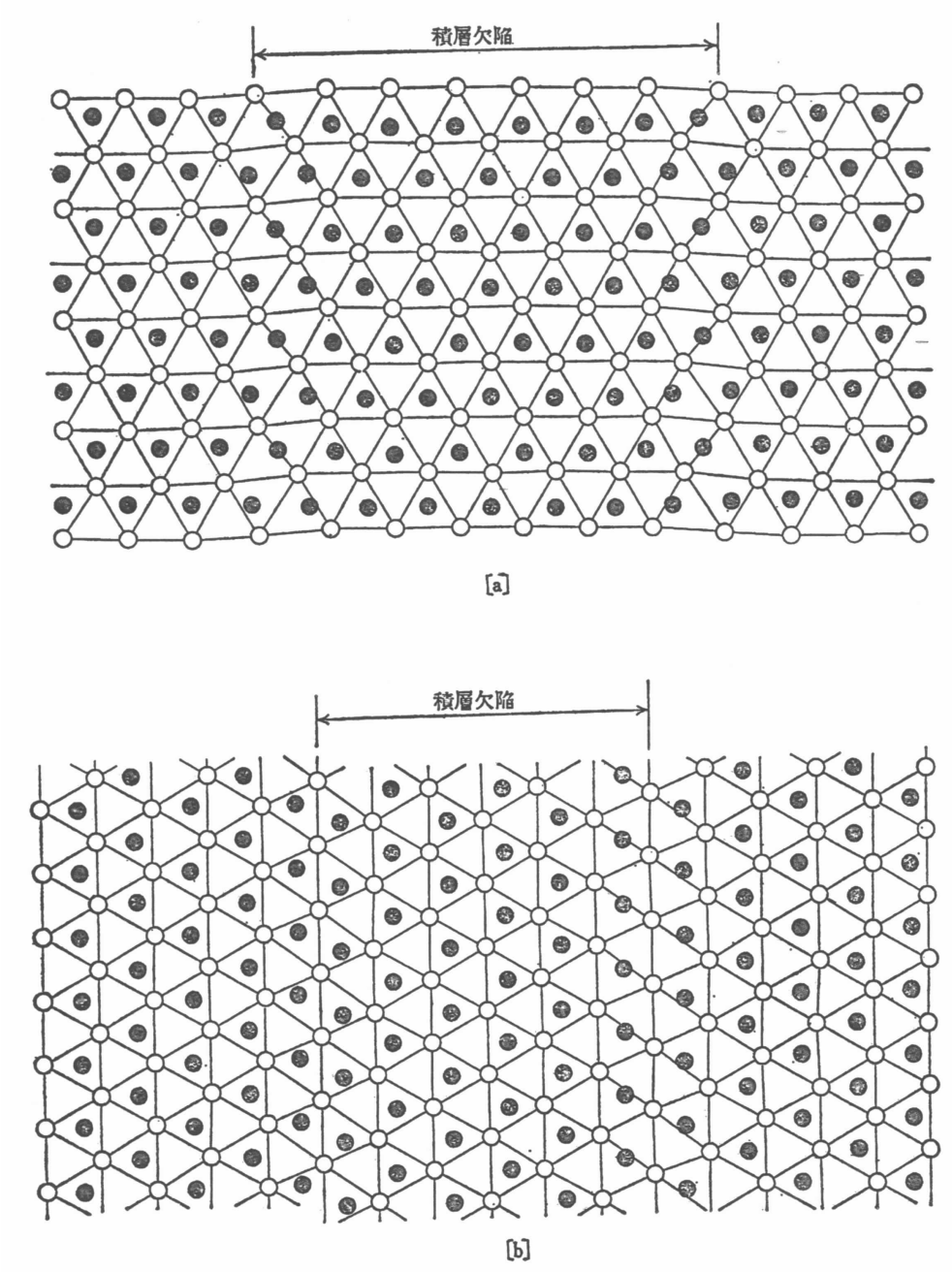


図 2.13: fcc の拡張転位．すべり面のすぐ上の原子を黒丸，すぐ下の原子を白丸で示す．〔a〕は刃状転位，〔b〕はらせん転位である [2].

またfccの刃状転位で部分転位が起こった場合，図2.14の様に完全転位ではaの層，bの層がそれぞれ1層ずつ抜けているだけであるが，図2.15の様に部分転位では同じ様にa層とb層が抜けてはいるが，抜けた間の部分ですべり面の上と下のa層とb層が反対の組に変わる．

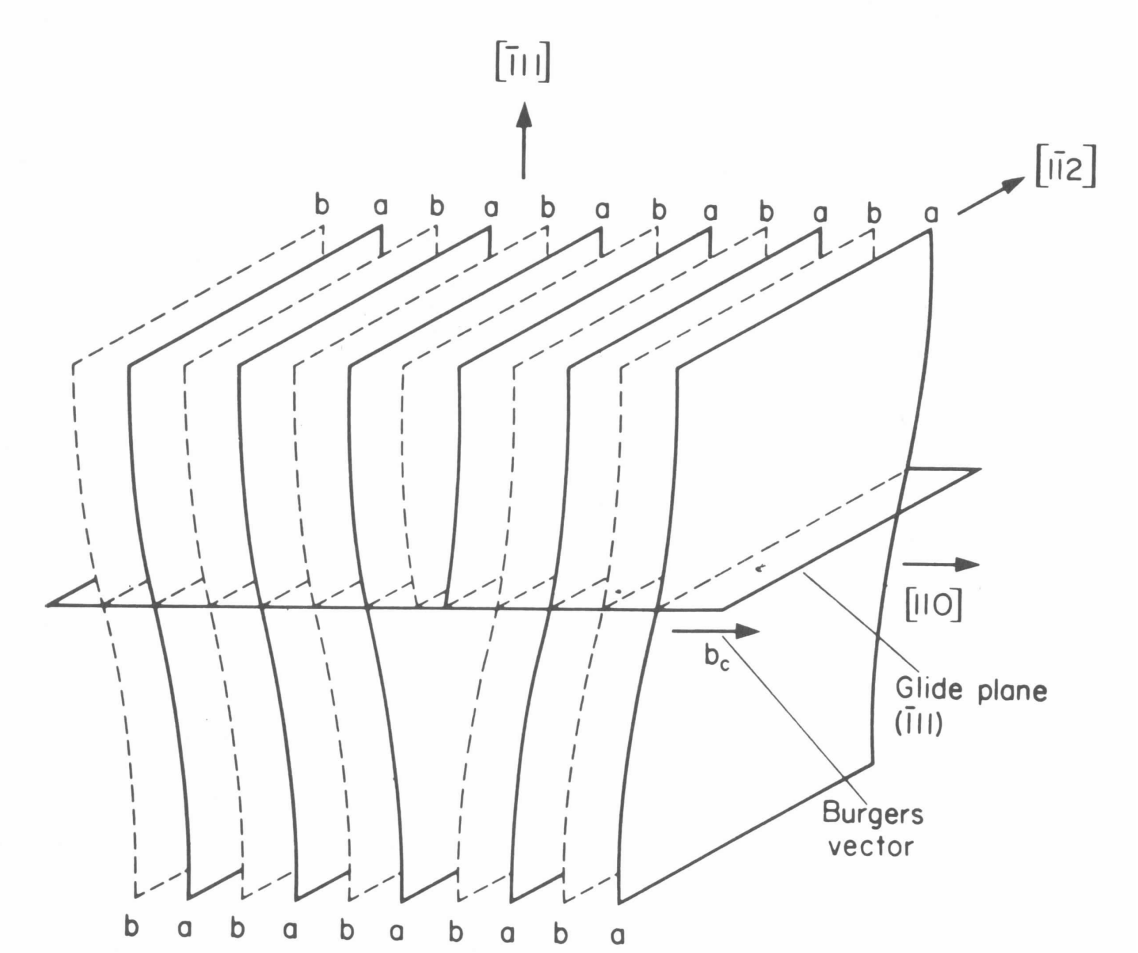


図 2.14: 完全転位の模式図 [3].

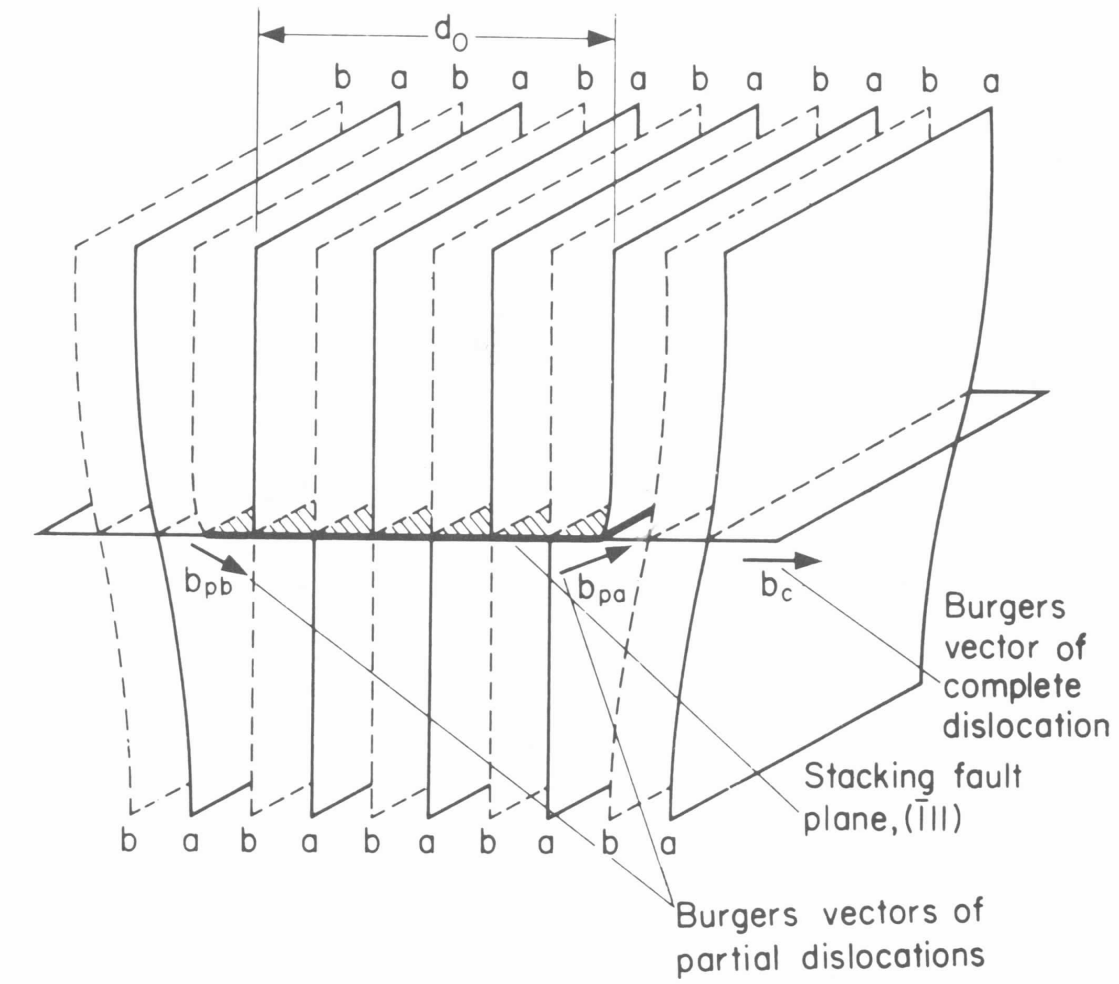


図 2.15: 部分転位の模式図 [3].

第3章　手法

物理モデルを視覚化させるシステムとして，[Ruby - MEL - Maya] を採用する．

3.1　Ruby

Ruby とはまつもとゆきひろ氏により開発されたオブジェクト指向スクリプト言語である．一般的に，Ruby は数値計算には向いていないと言われる．実際に，Ruby は C や Fortan などの他の言語と比べ，実行速度などの面で遅い．しかし，Ruby の言語は，単純な構文で書かれており，可読性に優れている．さらにコンパイルを必要としないインプリンタ方式を採用しているため，実行の手間が少ない．加えて，高機能なスクリーンエディタとして有名な Emacs と併用すれば，構文に応じてスクリプトが色分けされるので．開発環境が非常に良いものとなる．本研究では，MEL の複雑な計算環境に代わるインターフェースとして Ruby を選定した．

3.2　Maya

Maya は，オープンアーキテクチャを基板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いるプロ仕様のハイエンドソフトである．グラフィックス性に優れ，非常に高度な機能を有しているため，自由度も高い．また，Maya はその GUI の全てを MEL というスクリプト言語で実行可能となっている．本研究では，実際に視覚化を行うインターフェースとして Maya を選定した．

MEL： MEL(Maya Embedded Lanquage) とは Maya で使用できるスクリプト言語であり，Maya の GUI の機能の全てをまかなうことが可能となっている．MEL のみで CG の作成，カスタマイズを行うことができるが，Maya 専用のスクリプトエディタ上でしか実行できない．また，線形計算などの数値計算に向いておらず，構文も複雑な構成となっている．

3.3 Ruby - MEL - Maya

視覚化システム [Ruby - MEL - Maya] の実行の流れについて解説する．先に記したように，視覚化を実現させるツールとして，Ruby，Maya を使用した．まず，ユーザが数値計算を行うメインスクリプトを Ruby にて作成，実行する．そのデータを反映させた MEL スクリプトを出力させる．そして，Maya のインターフェースで生成した MEL スクリプトを読み込むことで視覚化を実現させる．また，この作業を効率良く行うため，視覚化の目的に応じてカスタマイズを行う関数のライブラリを作成した．

第4章　視覚化のシステム

本章では，[Ruby - MEL - Maya] のシステムの実装，使用法を解説する．

4.1　原子座標データ

本研究で，視覚化させる面はx軸方向を [110]，y軸方向を $[1\bar{1}2]$，z軸方向を $[\bar{1}11]$ ととるため，複雑で原子の並びを想像し難い．原子座標を求める上で，原子の並びを直感的に見るために，Maple を使用し図 4.1 のような図をもとに，Ruby で原子座標データのスクリプトを作製した．付録に図 4.1 を作成した際の Maple のコードを付してある．

Maple：　Maple は，数式処理，数値計算，グラフ作成などを行うソフトウェアの一つである．Maple を使うと，紙と鉛筆で行う数学の計算や作図をコンピュータで行うことができる．

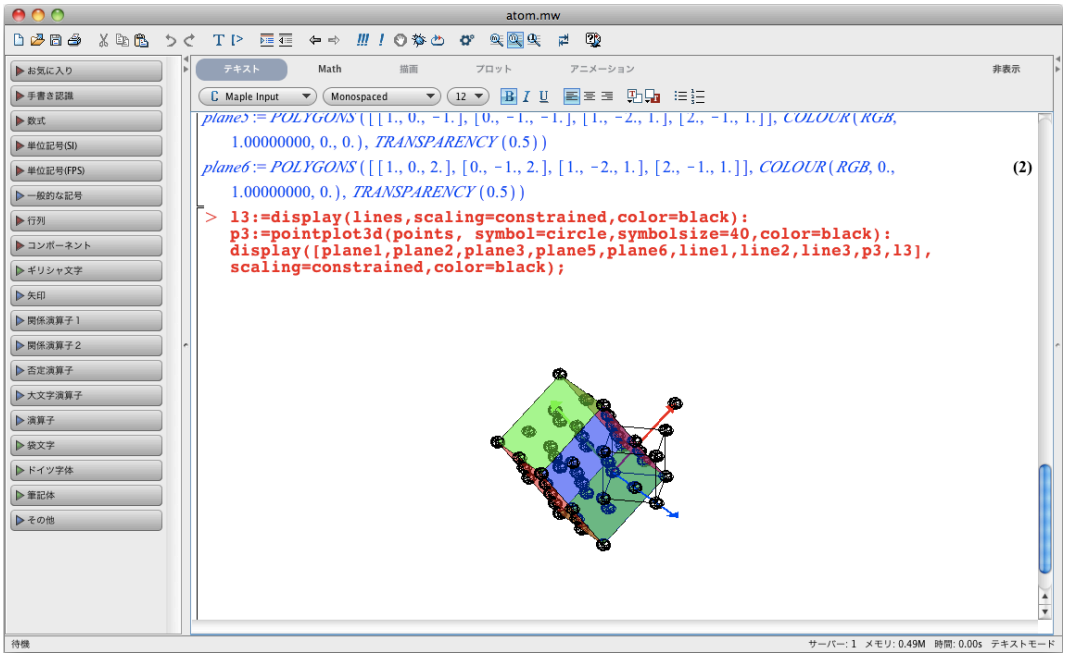


図 4.1: Maple で作成した原子モデル．

Maple で作成した図 4.1 をもとに，図 2.9 の立方体の一辺を 3 として求めた原子座標値を次に示す．

原子座標

```
a11=Vector[0.00000,0.00000,0.00000]
a13=Vector[2.12132,0.00000,0.00000]
a22=Vector[1.06066,-1.83712,0.00000]
a24=Vector[3.18198,-1.83712,0.00000]
a31=Vector[0.00000,-3.67422,0.00000]
a33=Vector[2.12132,-3.67422,0.00000]
a42=Vector[1.06066,-5.51133,0.00000]
a44=Vector[3.18198,-5.51133,0.00000]
b12=Vector[1.06066,-0.61237,2.12132]
b14=Vector[3.18198,-0.61237,2.12132]
b21=Vector[0.00000,-2.44948,2.12132]
b23=Vector[2.12132,-2.44948,2.12132]
b32=Vector[1.06066,-4.28659,2.12132]
b34=Vector[3.18198,-4.28659,2.12132]
b41=Vector[0.00000,-6.1237,2.12132]
b43=Vector[2.12132,-6.1237,2.12132]
c11=Vector[0.00000,-1.22474,4.24264]
c13=Vector[2.12132,-1.22474,4.24264]
c22=Vector[1.06066,-3.06185,4.24264]
c24=Vector[3.18198,-3.06185,4.24264]
c31=Vector[0.00000,-4.89496,4.24264]
c33=Vector[2.12132,-4.89496,4.24264]
c42=Vector[1.06066,-6.73607,4.24264]
c44=Vector[3.18198,-6.73607,4.24264]
```

これを Maya で表示すると次のようになる．

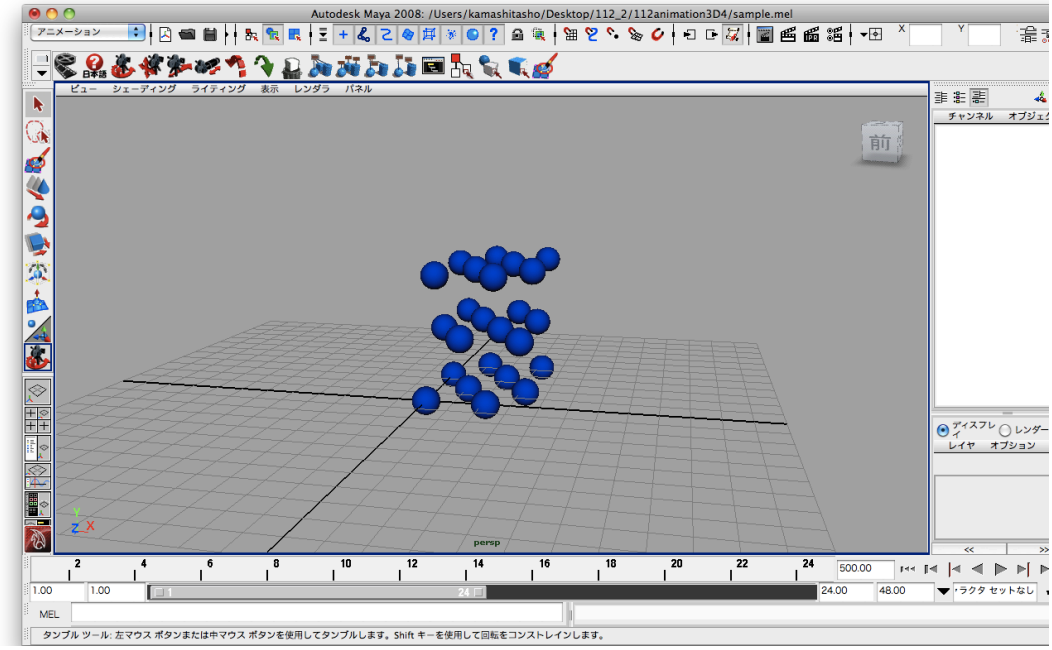


図 4.2: Maya での表示．

今回はこれを最小単位とし，for 文，if 文を用いて拡張させ表示を行った．

4.2 内部緩和

今回，転位を表す方法として，結晶内のある 1 層を抜き取り，内部緩和させることによって，転位の起こっている結晶を表現した．

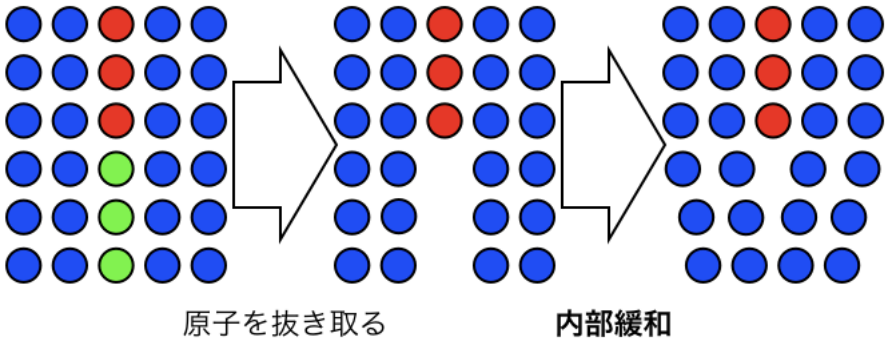


図 4.3: 内部緩和のイメージ．

この内部緩和を行うスクリプトは，西谷研究室，岩倉と松塚が作成した inner_relax.rb を使用した．コードは，付録に付してある．

4.3 静止画表示

静止画表示を行うスクリプト mel1.rb 内の関数について以下に示す.

read: 原子座標の入ったスクリプトを読み込む関数である. rm_pos.txt, atom_pos.txt, inner_relax.txt にはそれぞれ,

1. 抜き取る原子の原子座標
2. 1 以外の原子座標
3. 2 を内部緩和させた原子座標

が入っている.

Ruby 記述例

```
$read_count=0
def read
  rm_pos=[];atom_pos=[];inner_relax_pos=[]
  file1 = File.open('rm_pos.txt','r')
  file2 = File.open('atom_pos.txt','r')
  file3 = File.open('inner_relax_pos.txt','r')
  file = [file1,file2,file3]
  file.each do |i|
    $read_count+=1
    while line = i.gets do
      if (res=line.chomp.scanf(" [%f,%f,%f], "))!=[] then
        if    $read_count==1 then rm_pos << [res[0],res[1],res[2]]
        elsif $read_count==2 then atom_pos << [res[0],res[1],res[2]]
        elsif $read_count==3 then inner_relax_pos << [res[0],res[1],res[2]]
        end
      end
    end
    res=[]
  end
  return rm_pos,atom_pos,inner_relax_pos
end
```

color: 色を作成する関数である. MEL で色を表すスクリプトには lambert や blinn がある. lambert は最もよく使用される関数である. ライトが均一に拡散されるため光沢のない材質を表現できる. blinn は, オブジェクト表面に光沢やつやができるため金属などの材質に用いるとよい. 今回は, blinn を採用した. lambert, blinn は色を指定する RGB 値を引数とし, 色を設定する MEL スクリプトを返す.

Ruby 記述例

```
$color_blinn_count=1
def color(rgb)
  cc = $color_blinn_count
  FILE.printf("shadingNode -asShader blinn;\n")
  FILE.printf("sets -renderable true -noSurfaceShader true -empty -name blinn%dSG;\n",cc)
  FILE.printf("connectAttr -f blinn%d.outColor blinn%dSG.surfaceShader;\n",cc,cc)
  FILE.printf("select -r blinn%d;\n",cc)
  FILE.printf("setAttr \"blinn%d.color\" -type double3 %1.5f %1.5f %1.5f;\n"
                                                              ,cc,rgb[0],rgb[1],rgb[2])
  $color_blinn_count += 1
  tmp = 'blinn'+cc.to_s
  return tmp
end
```

background： 背景を表示する関数である．color を引数とし，背景となる球を作成する．

Ruby 記述例

```
def background(color)
  FILE.printf("string $%s[];\n",'background')
  FILE.printf("$%s = `%s`;\n",'background','sphere')
  FILE.printf("sets -e -forceElement %sSG;\n",color)
  FILE.printf("setAttr ($background[0] + \".sx\") 500;\n")
  FILE.printf("setAttr ($background[0] + \".sy\") 500;\n")
  FILE.printf("setAttr ($background[0] + \".sz\") 500;\n")
end
```

atom： 原子を表示する関数である．原子座標と color を引数とし，原子となる球を作成する．

Ruby 記述例

```
def atom(atom_pos,color)
  FILE.printf("global proc make_ball(float $x,float $y,float $z){\n")
  FILE.printf("sphere;\n")
  FILE.printf("scale 0.5 0.5 0.5;\n")
  FILE.printf("move $x $y $z;\n")
  atom_write1(color)
  atom_write2(i,atom_pos)
end
def atom_write1(color)
  FILE.printf("sets -e -forceElement %sSG;}\n",color)
end
def atom_write2(i,atom_pos)
  FILE.printf("make_ball(%7.5f,%7.5f,%7.5f);\n",atom_pos[i][0],atom_pos[i][2],atom_pos[i][1])
end
```

stick： ボンドを表示する関数である．原子座標と color を引数とし，ボンドとなる棒を作成する．関数 neighbor では，原子座標を引数とし，len の値を変更することによって第一近接の原子のボンドだけを指定することができる．関数 rotate_angle では，第一近接の原子同士の角度の計算を行っている．

Ruby 記述例

```
def stick(atom_pos,color)
  $pair = Array.new
  FILE.printf("global proc make_cylinder(float $x,float $y,float $z
                                                    ,float $len,float $yy,float $zz){\n")
  FILE.printf("cylinder;\n")
   stick_write1(color)
   for i in 0..atom_pos.size-1 do
     for j in 0..atom_pos.size-1 do
       if i==j then next end
       neighbor(atom_pos[i],atom_pos[j])
     end
   end
  stick_write2
end
def stick_write1(color)
  FILE.printf("scale $len 0.1 0.1;\n")
  FILE.printf("move $x $y $z;\n")
  FILE.printf("rotate 0 $yy $zz;\n")
  FILE.printf("sets -e -forceElement %sSG;}\n",color)
end
def stick_write2
  $pair.each do |v|
    ro = rotate_angle(v[0])
    FILE.printf("make_cylinder(%7.5f,%7.5f,%7.5f,%7.5f,%7.5f,%7.5f);\n"
                                      ,v[1][0],v[1][1],v[1][2],(v[0].r)*0.5,ro[1],ro[2])
  end
end
def neighbor(atom_pos1,atom_pos2)
  len = 3.67
  v1=atom_pos1
  a=Vector[v1[0],v1[2],v1[1]]
  v2=atom_pos2
  b=Vector[v2[0],v2[2],v2[1]]
  tmp_vector = a - b
  length = tmp_vector.r
  if(0<length && length<len)then
    if(tmp_vector[0]<0)then
      tmp_vector = tmp_vector*-1
    end
    $pair.push [tmp_vector, (a+b)*0.5]
  end
  $pair.uniq!
end
def rotate_angle(vec)
  x = vec[0]
  y = vec[1]
  z = vec[2]
  len = vec.r
  theta2 = -asin(z/len)
  tmp = y/(len*cos(theta2))
  if(-1.001<=tmp && tmp<-0.999)then
    theta3 = -1.5707963267949
  elsif(0.999<=tmp && tmp<1.001)then
    theta3 = 1.5707963267949
  else
    theta3 = asin(tmp)
  end
  yy = theta2*180/PI
  zz = theta3*180/PI
  return [0,yy,zz]
end
```

4.4 アニメーション表示

アニメーション表示を行うスクリプト mel2.rb 内の関数について以下に示す．静止画と同じ部分は割愛する [4].

anime： アニメーションを表示する関数である．

- オブジェクトをある時間内で現れたり消えたりするようなアニメーションを作成する場合．
  原子座標と color を引数とする．

Ruby 記述例

```
def anime1(rm_pos,color)
  FILE.printf("global proc ball_color1(float $x1,float $y1,float $z1){\n")
  FILE.printf("$name = ‘sphere‘;")
  FILE.printf("scale 0.5 0.5 0.5;\n")
  FILE.printf("setAttr($name[0] + \".tx\") $x1;\n")
  FILE.printf("setAttr($name[0] + \".ty\") $y1;\n")
  FILE.printf("setAttr($name[0] + \".tz\") $z1;\n")
  FILE.printf("sets -e -forceElement %sSG;\n",color)
  FILE.printf("\n")
  FILE.printf("currentTime 1;\n")
  FILE.printf("setAttr blinn1.transparencyR 0.0;\n")
  FILE.printf("setAttr blinn1.transparencyG 1.0;\n")
  FILE.printf("setAttr blinn1.transparencyB 0.0;\n")
  FILE.printf("setKeyframe -at \"transparencyR\" blinn1;\n")
  FILE.printf("setKeyframe -at \"transparencyG\" blinn1;\n")
  FILE.printf("setKeyframe -at \"transparencyB\" blinn1;\n")
  FILE.printf("currentTime 50;\n")
  FILE.printf("setAttr blinn1.transparencyR 1;\n")
  FILE.printf("setAttr blinn1.transparencyG 1;\n")
  FILE.printf("setAttr blinn1.transparencyB 1;\n")
  FILE.printf("setKeyframe -at \"transparencyR\" blinn1;\n")
  FILE.printf("setKeyframe -at \"transparencyG\" blinn1;\n")
  FILE.printf("setKeyframe -at \"transparencyB\" blinn1;\n")
  FILE.printf("}\n")
  for i in 0..rm_pos.size-1 do
    FILE.printf("ball_color1(%7.5f,%7.5f,%7.5f);\n"
                ,rm_pos[i][0],rm_pos[i][2],rm_pos[i][1])
  end
end
```

- オブジェクトをある時間内で移動させるようなアニメーションを作成する場合.
  移動前の原子座標と移動後の原子座標と color を引数とする.

Ruby 記述例

```
def anime(atom_pos,inner_relax_pos,color)
  FILE.printf("global proc ball_move_color(float $x1,float $y1,float $z1
                                          ,float $x2,float $y2,float $z2){\n")
  FILE.printf("$name = ‘sphere‘;")
  FILE.printf("scale 0.5 0.5 0.5;\n")
  FILE.printf("setAttr($name[0] + \".tx\") $x1;\n")
  FILE.printf("setAttr($name[0] + \".ty\") $y1;\n")
  FILE.printf("setAttr($name[0] + \".tz\") $z1;\n")
  FILE.printf("sets -e -forceElement %sSG;\n",color)
  FILE.printf("\n")
  FILE.printf("currentTime 1;\n")
  FILE.printf("setAttr ($name[0] + \".tx\") $x1;\n")
  FILE.printf("setKeyframe -at \"tx\" $name[0];\n")
  FILE.printf("setAttr ($name[0] + \".ty\") $y1;\n")
  FILE.printf("setKeyframe -at \"ty\" $name[0];\n")
  FILE.printf("setAttr ($name[0] + \".tz\") $z1;\n")
  FILE.printf("setKeyframe -at \"tz\" $name[0];\n")
  FILE.printf("currentTime 100;\n")
  FILE.printf("setAttr ($name[0] + \".tx\") $x2;\n")
  FILE.printf("setKeyframe -at \"tx\" $name[0];\n")
  FILE.printf("setAttr ($name[0] + \".ty\") $y2;\n")
  FILE.printf("setKeyframe -at \"ty\" $name[0];\n")
  FILE.printf("setAttr ($name[0] + \".tz\") $z2;\n")
  FILE.printf("setKeyframe -at \"tz\" $name[0];\n")
  FILE.printf("}\n")
  for i in 0..atom_pos.size-1 do
    FILE.printf("ball_move_color(%7.5f,%7.5f,%7.5f,%7.5f,%7.5f,%7.5f);\n"
                ,atom_pos[i][0],atom_pos[i][2],atom_pos[i][1]
                ,inner_relax_pos[i][0],inner_relax_pos[i][2],inner_relax_pos[i][1])
  end
end
```

ターミナルでの Ruby スクリプトの実行方法を以下に示す.

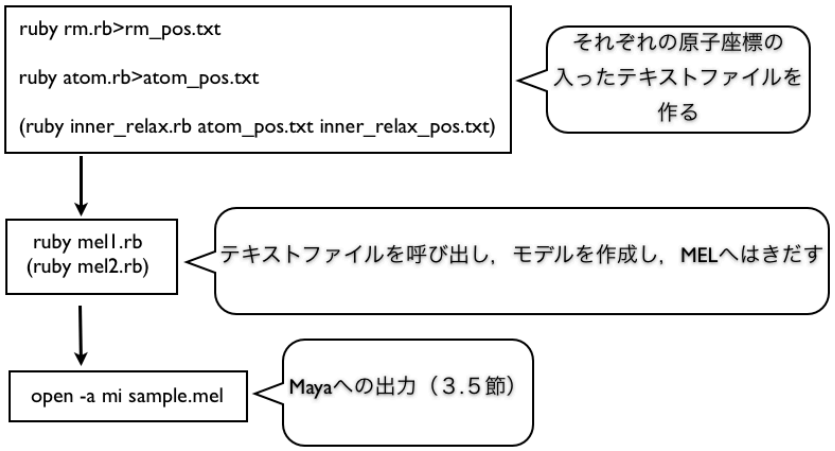


図 4.4: 実行方法

4.5 MELの実行

Ruby スクリプトから書き出された MEL スクリプトの実行．以下に Maya への出力方法を示す [5].

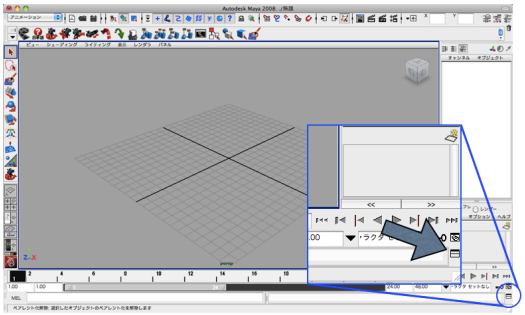


図 4.5: Maya インターフェース．

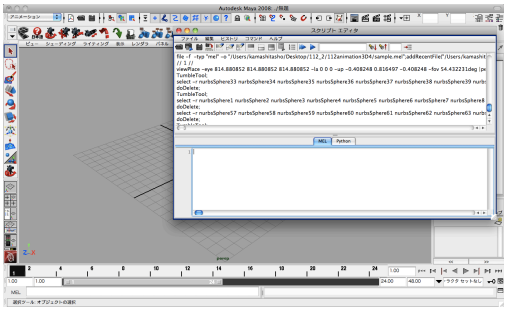


図 4.6: Maya のスクリプトエディタ.

MEL スクリプトの出力には，まず Maya のスクリプトを開く必要がある．図 4.5 の矢印の箇所をクリックすると図 4.6 にある Maya のスクリプトエディタが表示される．

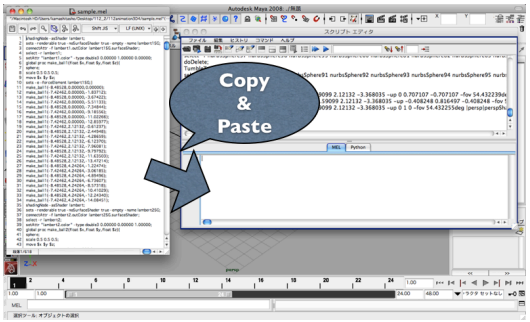


図 4.7: MEL の Copy&Paste.

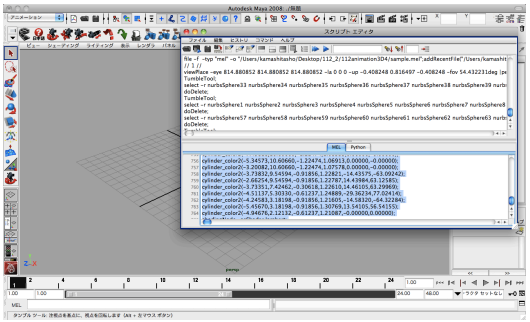


図 4.8: MEL の実行.

図 4.7 のように ’mi ’などのテキストエディタで開いた MEL スクリプトをスクリプトエディタに貼付ける．MEL スクリプトを図 4.8 のように全選択した状態で，キーボードより ’Control + enter ’を入力する．

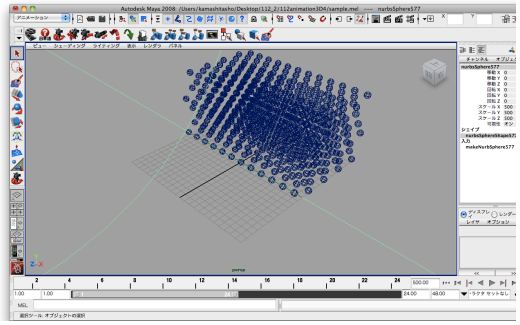


図 4.9: 出力結果（線表示）.

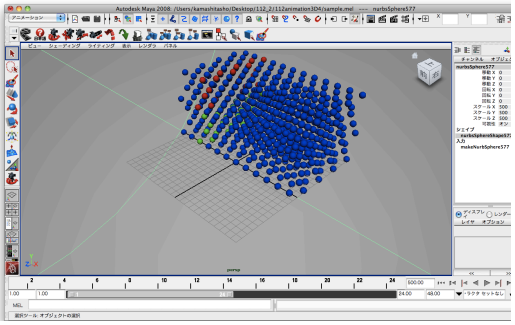


図 4.10: 出力結果（カラー表示）.

最終的に図 4.9，図 4.10 のような出力が得られる．オブジェクトの線表示とカラー表示はキーボードの ’5 ’のキーで変更が可能となっている．

4.6　アニメーションの保存方法

Maya で作成したアニメーションを QuickTime Player で見れる形式に保存する．その方法を以下に記述する．

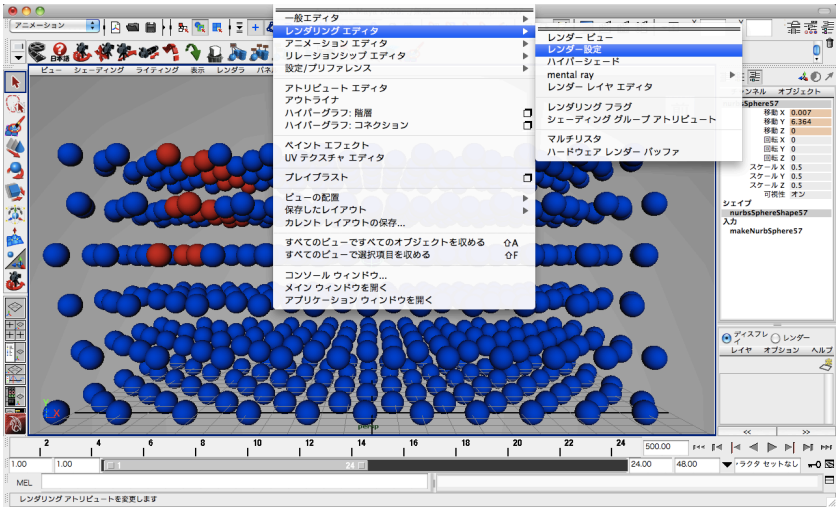


図 4.11: Maya インターフェース（ウインドウ）．

画面上部のウインドウからレンダリングエディタ／レンダー設定を選択．

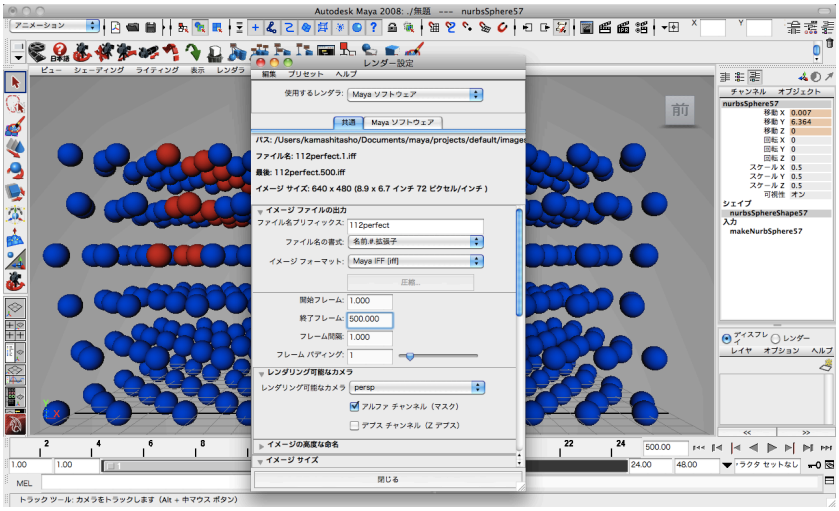


図 4.12: Maya インターフェース（レンダー設定／共通）．

ファイル名プリフィックスに適当な名前を入力．ファイル名の書式で名前.#.拡張子を選択．イメージフォーマットで Maya IFF [iff] を選択．開始フレームと終了フレームを指定する．レンダリング可能なカメラを選択（カメラを作製していない場合は，persp を選択）．

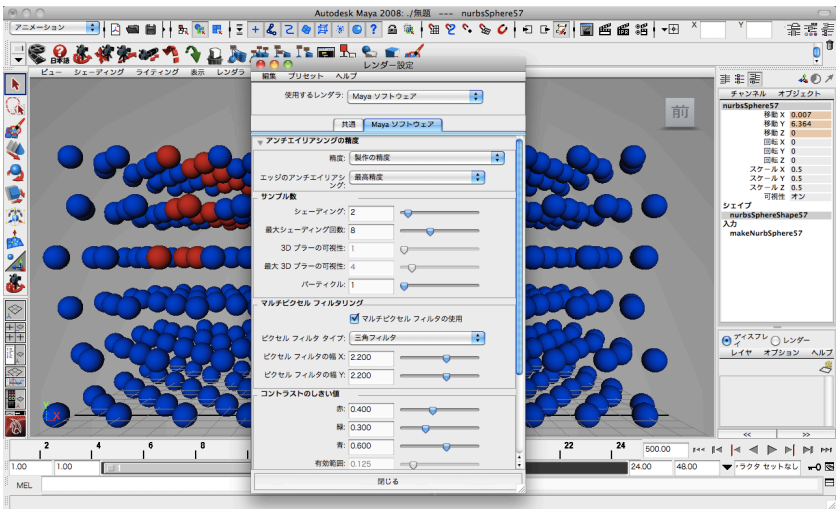


図 4.13: Maya インターフェース（レンダー設定／ Maya ソフトウェア）．

精度で製作の精度を選択すると，エッジのアンチエイリアシングが最高の精度に変わる．下部の閉じるをクリック．

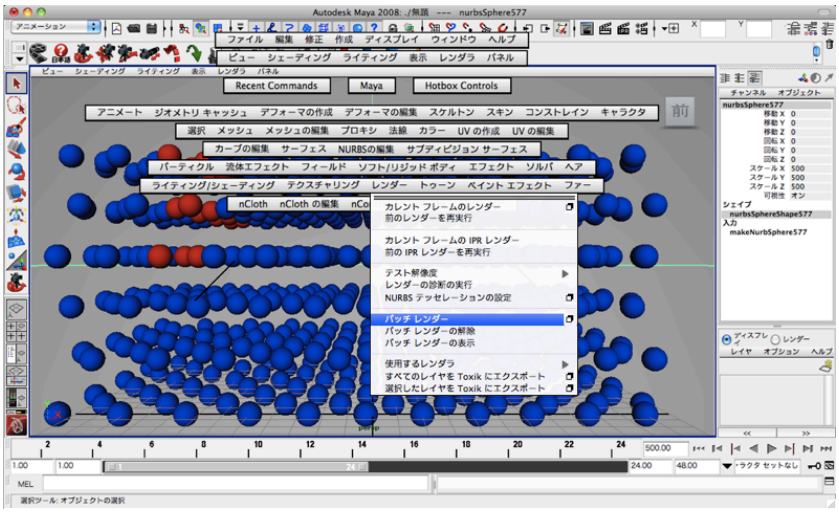


図 4.14: Maya インターフェース（レンダー）．

次に Maya の画面上で space キーを押す．出てくるウインドウのレンダー／バッチ　レンダーをクリック．するとレンダリングが開始される．

その際，下の Dock に Batch Render Monitor が起動している．これは強制終了させておくのがよい．

ファイル／シーケンスの表示を選択．先ほど決めたファイル名を入力すると，進行状況が確認できる．

最後まで終了したら，ファイル名.1.iff を開く．
Fcheck が起動して，アニメーションが表示される．

もし，MEL スクリプトの出力の際に，ターミナル上で open ファイル名.mel，と入力し Maya を開いた場合は，Fcheck のウインドウには何も表示されなくなる．

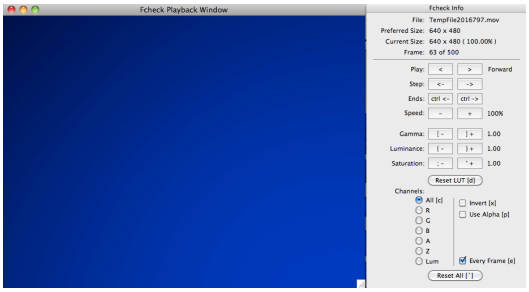


図 4.15: Fcheck（エラー時）．

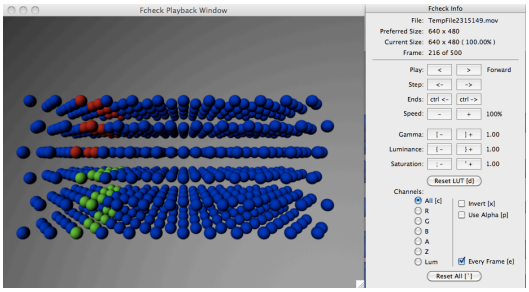


図 4.16: Fcheck（正常時）．

Fcheck の File から Save As Movie を選択．Use Transformations をクリックすると，QuickTime で見れる動画が作製される．

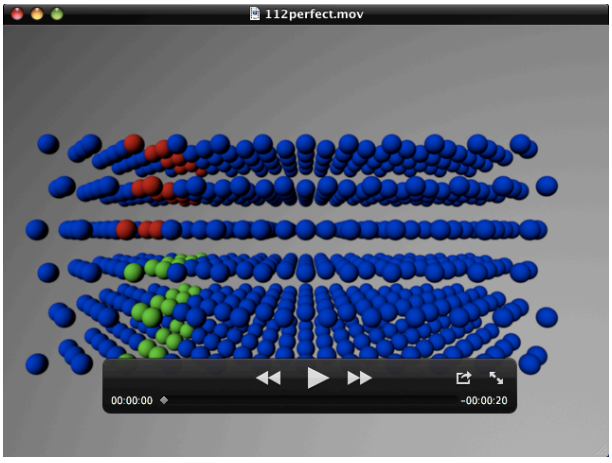


図 4.17: QuickTime Player での表示．

第5章　転位モデルの視覚化

5.1　刃状転位

刃状転位の表示結果を以下に示す.

まず図5.1のような完全結晶のモデルを作成した．図4.2で示したモデルをx軸（[110]）方向に6倍，y軸（[1$\bar{1}$2]）方向に2倍，z軸（[$\bar{1}$11]）方向に2倍拡張させ，[$\bar{1}$11]面を下から6層積んだ構造となっている．図2.14と対応させた場合，図5.1では，下から3層目と4層目の間がすべり面（glide plane）となる．図2.14において，すべり面の下でb，aの2層が抜けていることがわかる．図5.1では，抜ける原子の2層を緑色で表し，また原子が抜けた際に，すべり面をはさんだ上部の2層をextra half planeとし，赤色で表した.

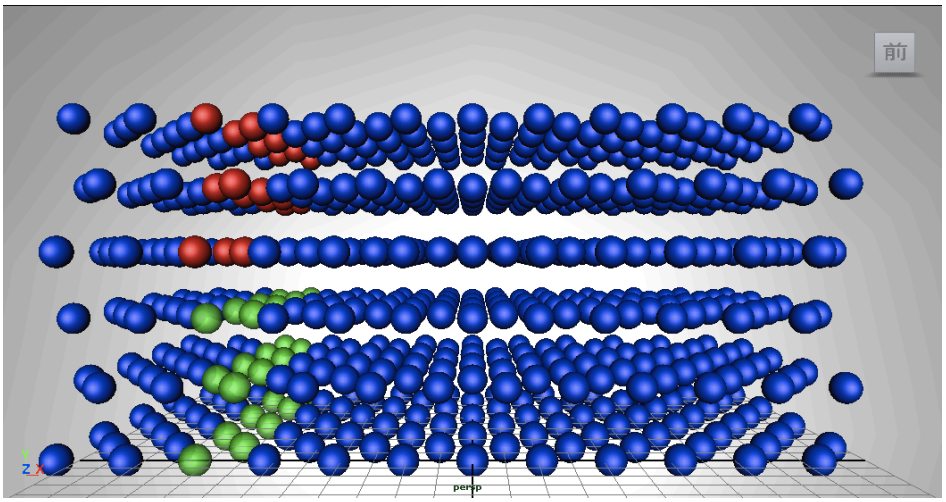


図 5.1: 刃状転位モデルの視覚化1（[1$\bar{1}$2]方向から）.

図 5.1 の結晶から緑色の原子を抜き取った状態を図 5.2 に示す．2.2 節で述べたように，原子がすべり，原子間に空隙が生まれることと原子を抜き取り空隙が生まれることは同様のことと考えられる．ここでは原子を抜き取ることで刃状転位を表現する．

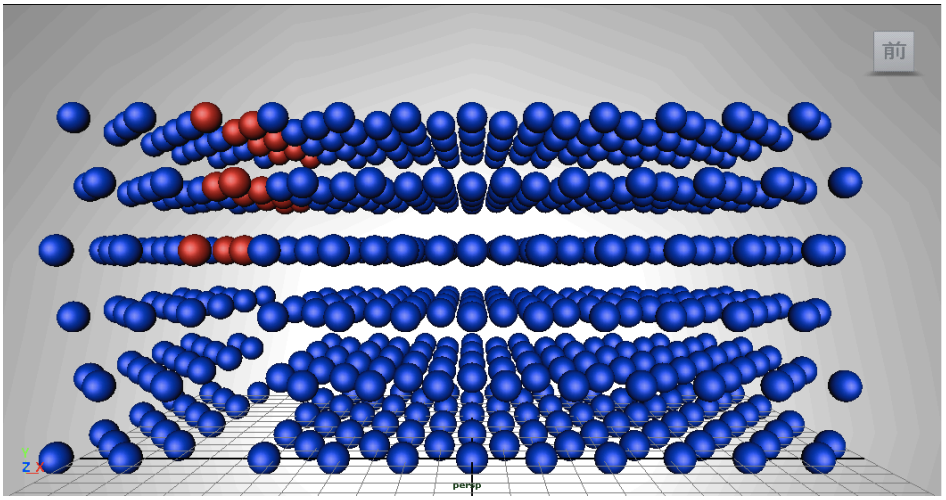


図 5.2: 刃状転位モデルの視覚化 2（$[1\bar{1}2]$ 方向から）．

図 5.2 のモデルの原子が抜けている部分の回りを囲うように赤線でバーガース・サーキットを表示した状態を図 5.3 に示す．囲めていない部分に空隙があり，すなわち転位が入っていることが確認できる．またその囲めていない部分にバーガース・ベクトル b を表すことができる．

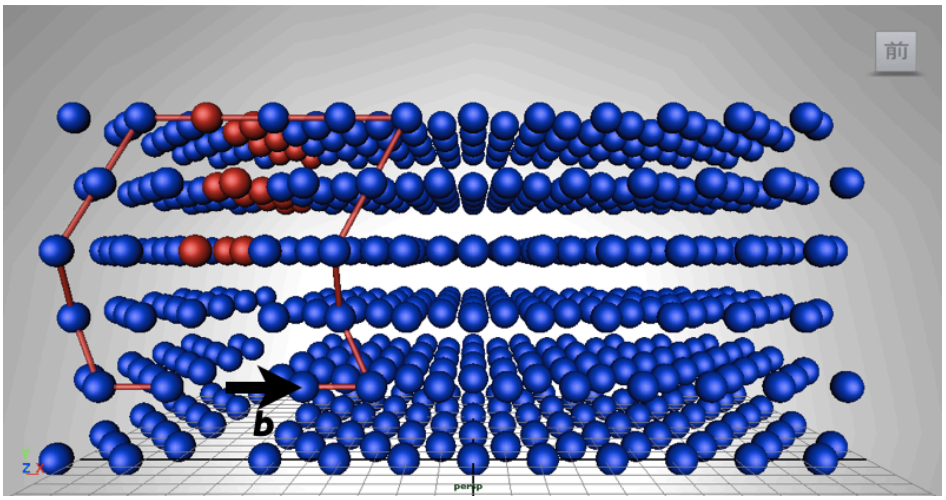


図 5.3: 刃状転位モデルの視覚化 3（$[1\bar{1}2]$ 方向から）．

図 5.2 の結晶を内部緩和させた状態を図 5.4 に示す．原子が抜けているところが互いに近づき，緩和された様子が確認できた．緩和されると一見転位が入ってないように見える．

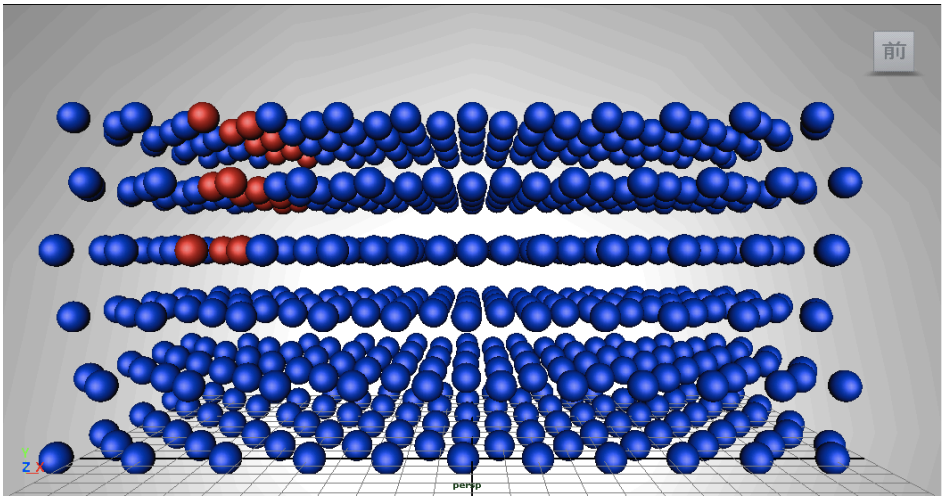


図 5.4: 刃状転位モデルの視覚化 4（$[1\bar{1}2]$ 方向から）．

図 5.4 に赤線でバーガース・サーキットを表示した状態を図 5.5 に示す．緩和されると，原子が抜けていた部分の原子間距離が近くなり，サーキットは囲むことができる．しかし，サーキットの上辺と下辺においてそれぞれに含まれる原子の数が異なり，転位が入っていることがわかる．バーガース・サーキットである領域を囲うことによって，転位を見つけ出すことができるという定義をこの図からも確認できる結果となった．

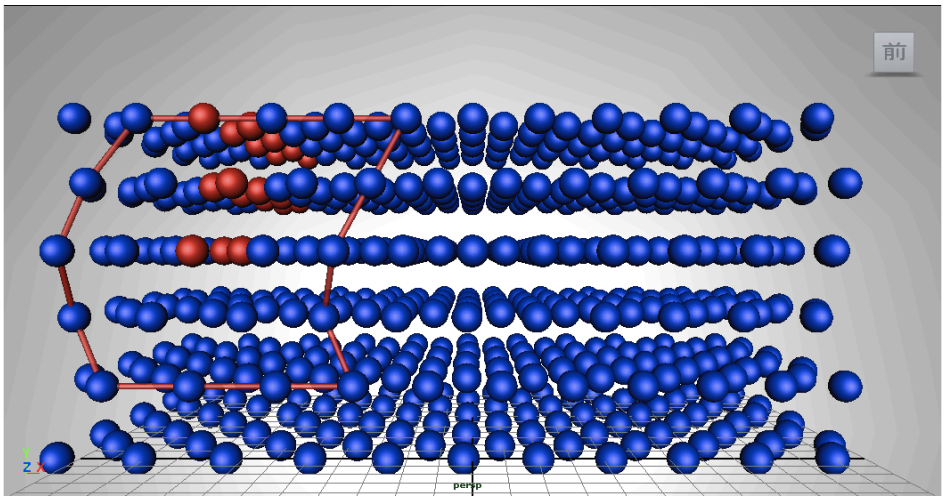


図 5.5: 刃状転位モデルの視覚化 5（$[1\bar{1}2]$ 方向から）．

1 番下の層だけを抜き取り，内部緩和前と内部緩和後の $[\bar{1}11]$ 方向から見た様子を比較した．原子が抜けたところの左右が，互い近づききれいに緩和されている様子が確認できた．

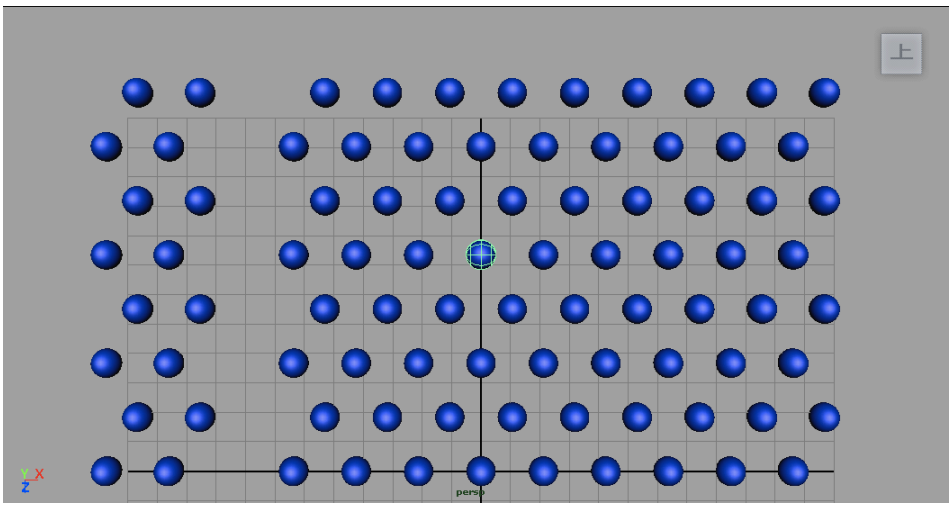


図 5.6: 内部緩和前（$[\bar{1}11]$ 方向から）．

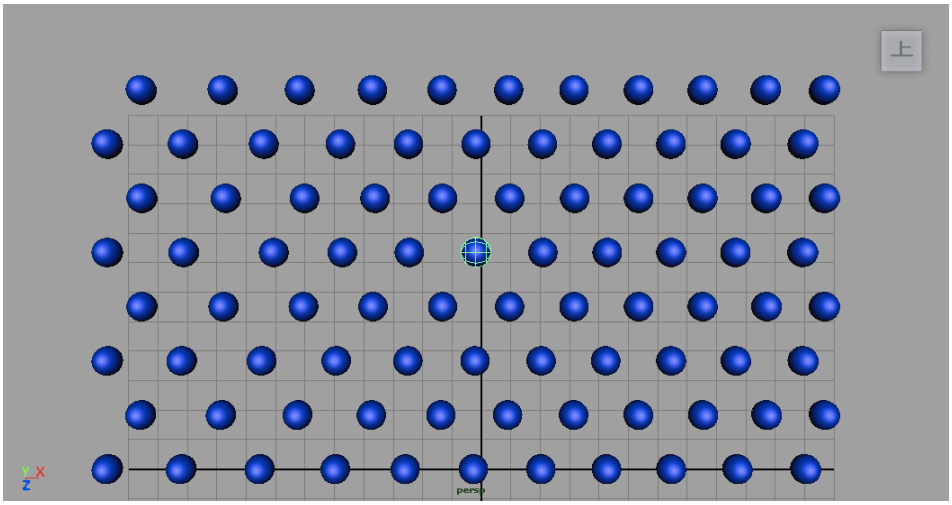


図 5.7: 内部緩和後（$[\bar{1}11]$ 方向から）．

extra half plane やバーガース・サーキットを色を変えて表示してみることによって，対称傾角粒界で等間隔に現れる転位を，原子レベルのシミュレーションから理解するときも，視覚的観点から補うことが可能となり，研究効率が高まった．

5.2　らせん転位

らせん転位の表示結果を以下に示す.

まず図 5.8 のような刃状転位の場合と同様の完全結晶のモデルを作成した．らせん転位は，図 2.3 で示されるように，転位（すべった部分とすべらない部分の境界）とバーガース・ベクトルが平行な転位であるため，ここでは四角で囲まれた部分を奥にずらすことでらせん転位を表現することとした.

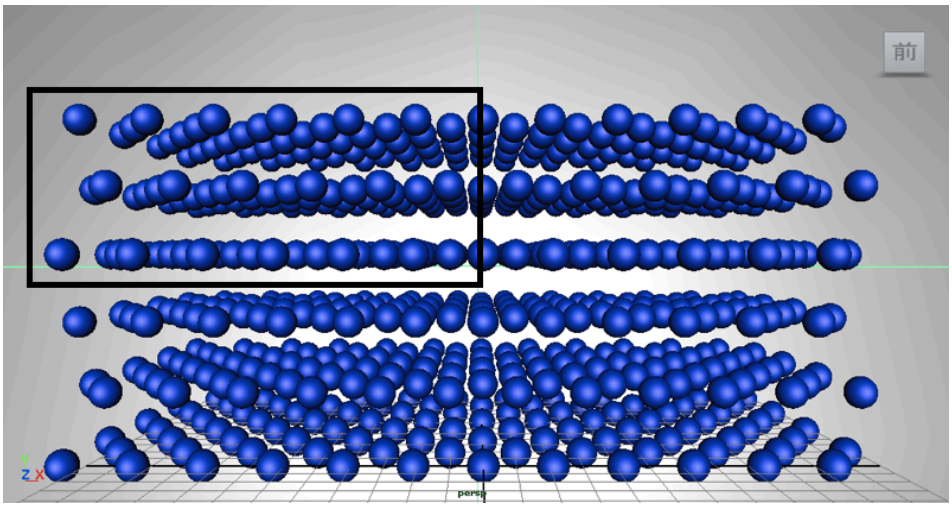


図 5.8: らせん転位モデルの視覚化 1（$[1\bar{1}2]$ 方向から）.

$[1\bar{1}2]$ 方向から見た転位が入る前と後の状態を比較する．図 2.8 をもとに赤線でバーガースサーキットを表示した．

図 5.9 は転位が入る前なので，図 2.8〔b〕のようにサーキットはきれいに囲むことができる．

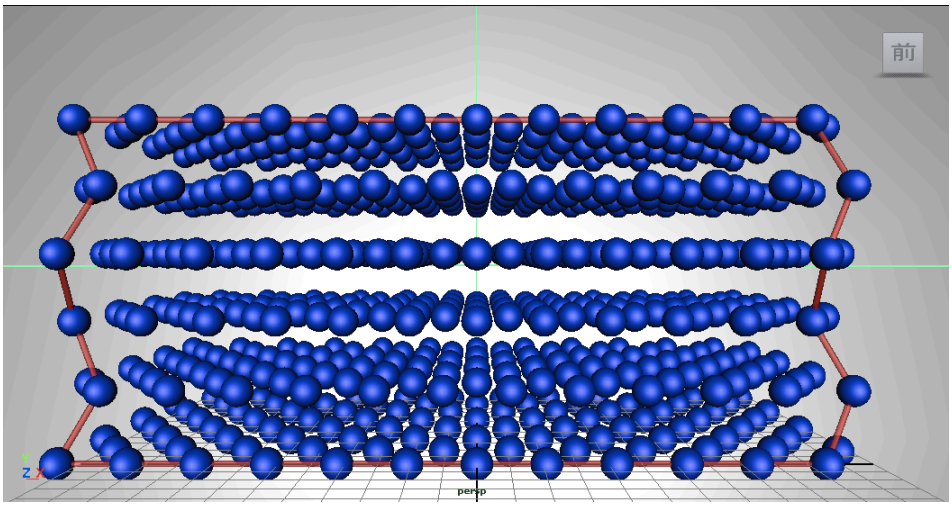


図 5.9: らせん転位モデルの視覚化 2（$[1\bar{1}2]$ 方向から）．

図 5.10 のように転位が入ると，図 5.8 の四角で囲まれた部分が奥にずれているため，図 2.8〔a〕のようにサーキット上の原子が，同一層上にないためサーキットを囲むことができない．囲めていない部分にバーガース・ベクトル b を表すことができる．ここで矢印は図の手前から奥の方向を向いている．

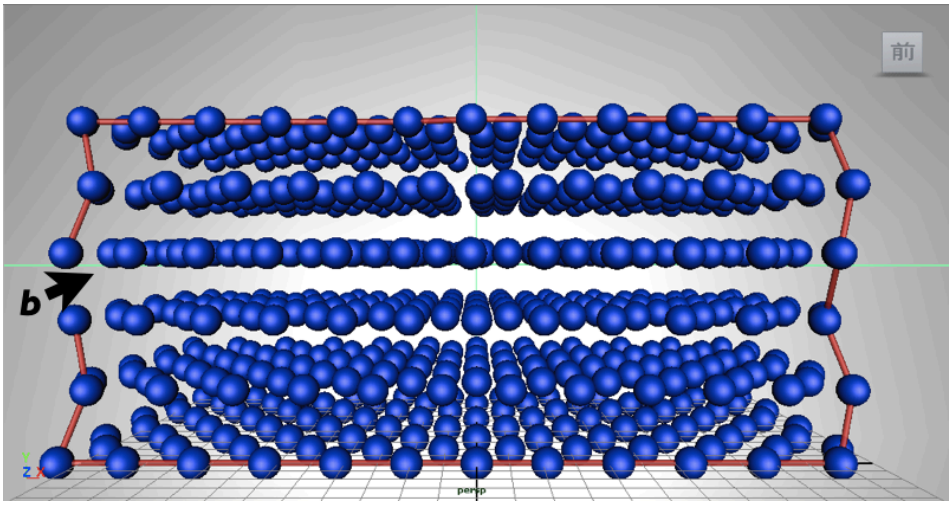


図 5.10: らせん転位モデルの視覚化 3（$[1\bar{1}2]$ 方向から）．

[110] 方向から見た転位が入る前と後の状態を比較する．こちらも図 2.8 をもとに赤線でバーガースサーキットを表示した．

完全結晶では，もちろん原子のずれは生じておらず，図 2.8〔b〕のようにサーキット上の原子はすべて同じ層にあることが確認できる．

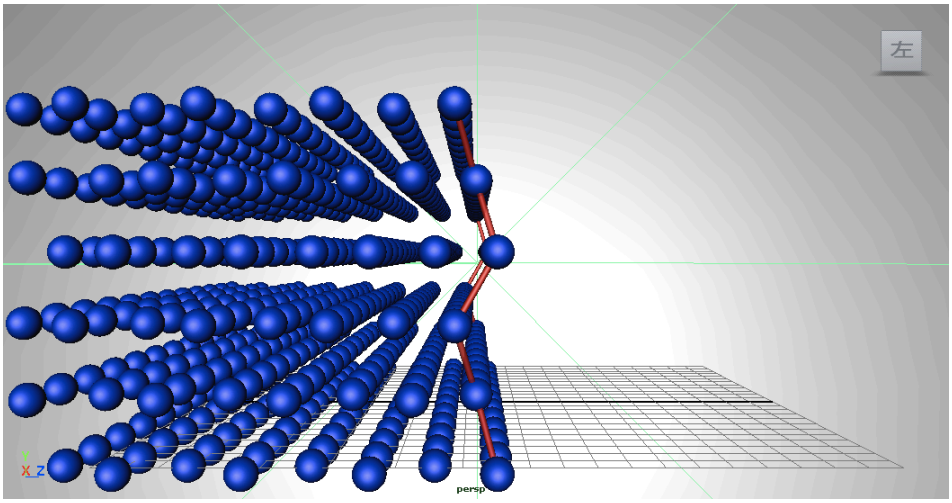


図 5.11: らせん転位モデルの視覚化 4（[110] 方向から）．

図 5.9 と図 5.10 の比較では確認しずらかった奥へのずれの確認が可能となった．図 5.10 と同様に囲めていない部分にバーガース・ベクトル b を表すことができ，ずれる方向を直感的に見ることが可能となり，らせん転位を理解し易くなる結果となった．

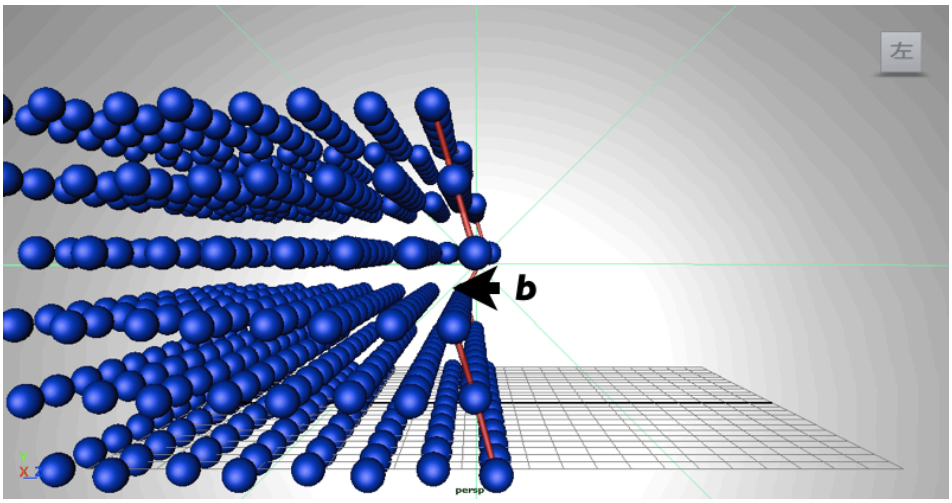


図 5.12: らせん転位モデルの視覚化 5（[110] 方向から）．

第6章　積層欠陥，部分転位モデルの視覚化

6.1　積層欠陥

第 2 章の 2.4 節で述べた積層欠陥が起こっているモデルを表現した．

図 6.1 は図 5.1 のような fcc の完全結晶モデルを [110] 方向から見たモデルである．1 番下の層を A 層とした場合，完全結晶では，下から…A B C A B C…のような順で層が並んでいることが確認できる．

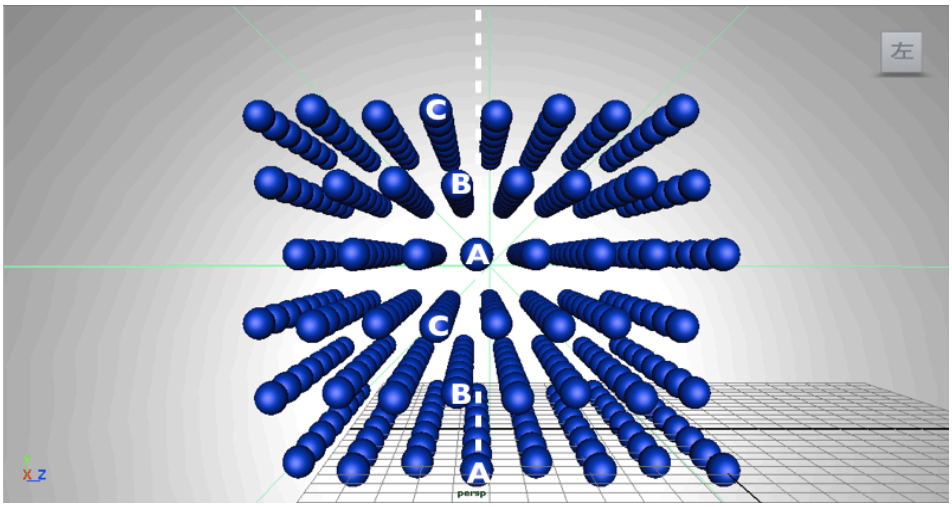


図 6.1: fcc の完全結晶モデル（[110] 方向から）．

積層欠陥は，図 6.2 の$\overrightarrow{BC}$，$\overrightarrow{CB}$のように原子がすべるときに起こる．よって [110] 方向で見た際に，図 6.1 の下 3 層を図 6.2 の CB’ 分の距離左へ B’B 分手前へずらすことで表現可能であると考えた．

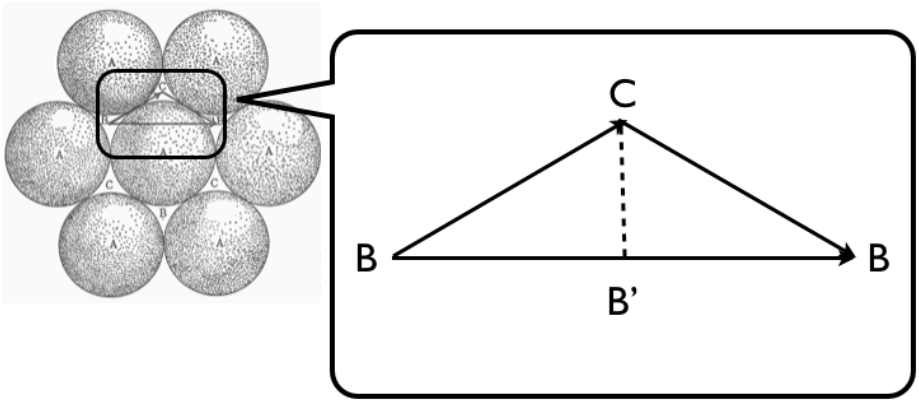


図 6.2: fcc のすべりにおける原子移動過程の拡大図．

図 6.3 は fcc の積層欠陥モデルを [110] 方向から見たモデルである．図 6.2 の CB’，B’B 分原子をずらしたことで，こちらも 1 番下の層を A とした場合，完全結晶では下から…A B C A B C…の順であったのに対して，積層欠陥中では…A B C B C A…のような順で層が並び，完全結晶の場合とは変化していることを確認することが可能となった．

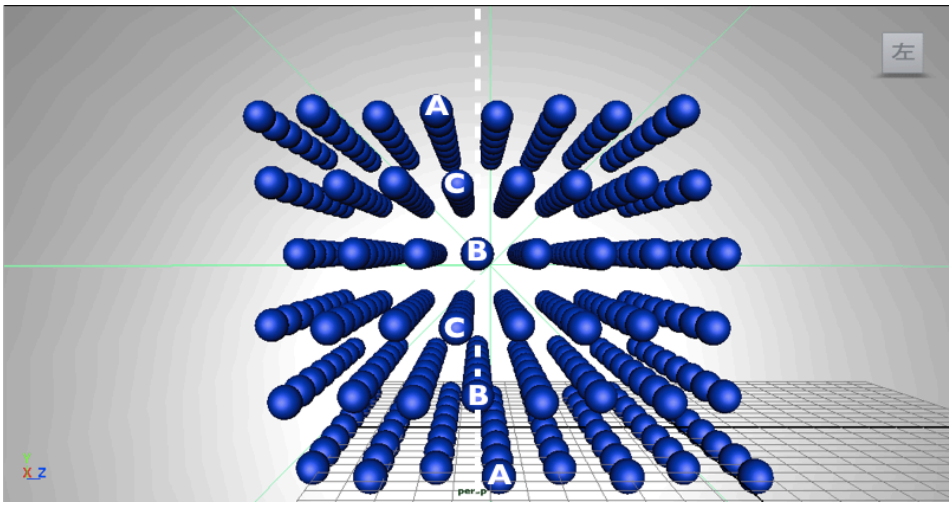


図 6.3: fcc の積層欠陥モデル（[110] 方向から）．

6.2　部分転位

第 2 章の 2.5 節で述べた部分転位が入っているモデルを表現した．

今度は，図 6.1，図 6.3 のモデルを [$\bar{1}$11] 方向から見た．１番下の層だけを表示すると，それぞれ図 6.4，図 6.5 のようになった．図 6.4 のバーガースベクトル$\overrightarrow{BB}$が図 6.5 の$\overrightarrow{BC}$，$\overrightarrow{CB}$のように分解されたように転位が入り，その間の部分で積層欠陥が起こっていることが確認できた．

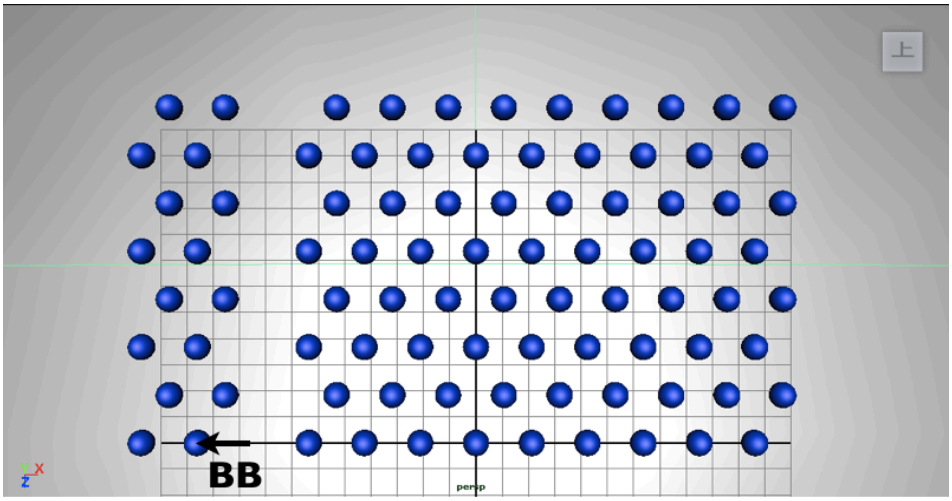


図 6.4: fcc の完全転位モデル（[$\bar{1}$11] 方向から）．

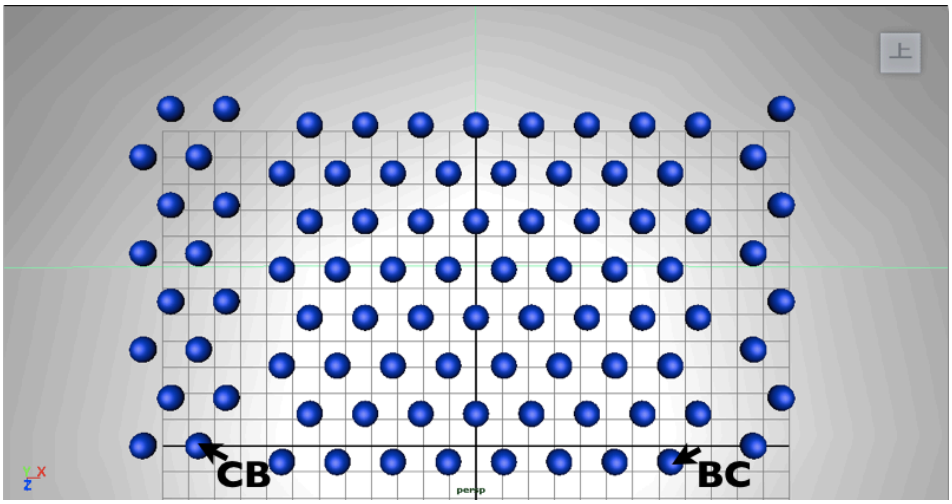
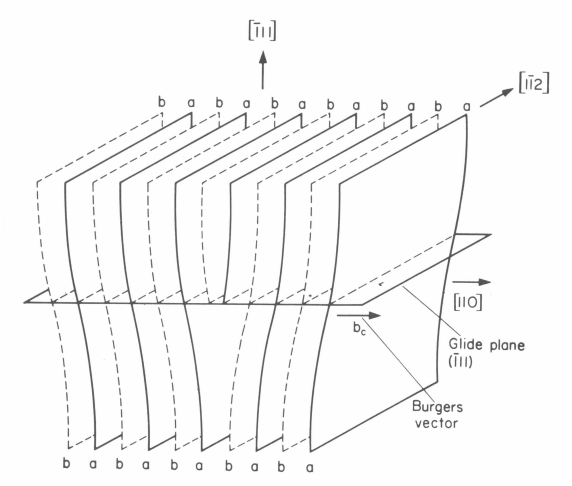


図 6.5: fcc の部分転位モデル（[$\bar{1}$11] 方向から）．

第 2 章の 2.5 節でも述べたが，結晶中に部分転位が入ったとき図 6.6，図 6.7 のようにすべり面の上下で a 層と b 層が反転することがわかっている．



,

図 6.6: 完全転位の模式図.

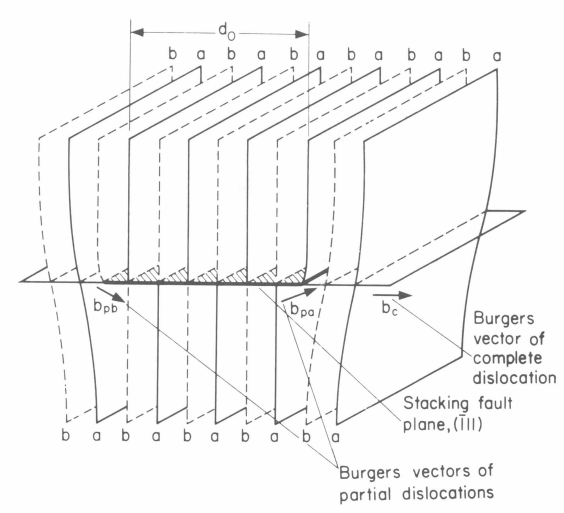


図 6.7: 部分転位の模式図.

下から３層目と４層目（すべり面のすぐ下とすぐ上）の層をそれぞれ白と黒の原子で表現したモデルを次に示す．完全結晶では，すべり面の下（白色）の部分で a，b 層が隣同士で抜けたようになっているため，すべり面の上下で a，b の並びは変わらない．

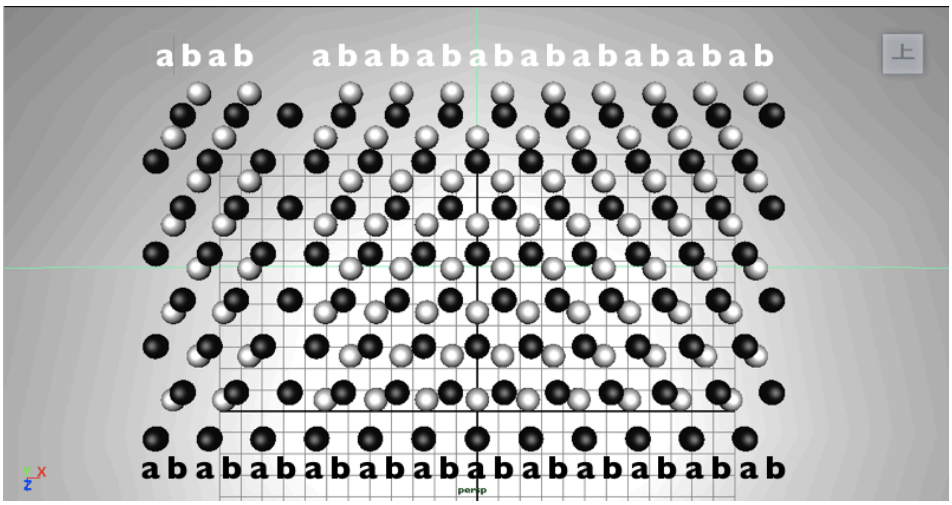


図 6.8: fcc の完全転位モデル（[$\bar{1}$11] 方向から）．

一方部分転位では，すべり面の下（白色）の部分で図 6.5 のように原子がすべったため，a，b 層が離れた部分で抜けたようになり，模式図通りすべり面の上下で a，b の並びが反転するということが確認できた．

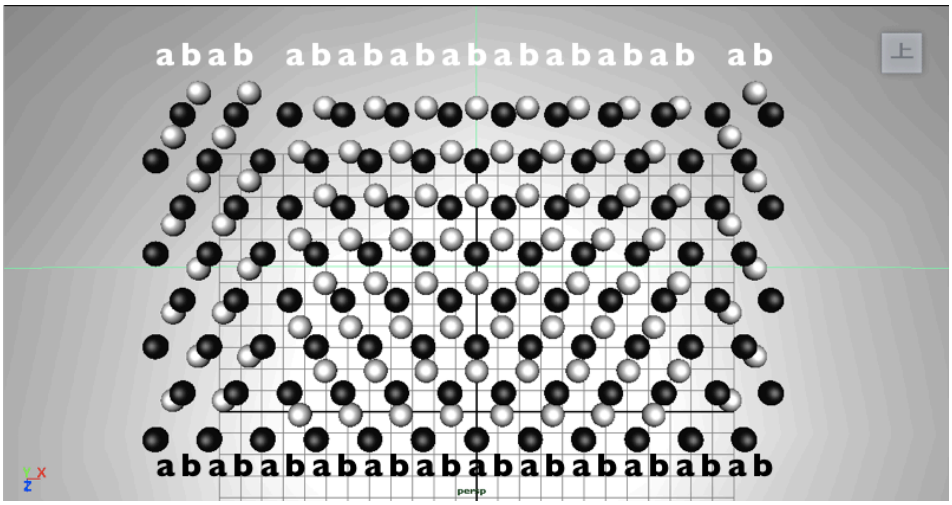


図 6.9: fcc の部分転位モデル（[$\bar{1}$11] 方向から）．

最後に部分転位モデルの内部緩和前と内部緩和後のモデルを以下に示す．extra half plane を赤色で表示した．アニメーションでは，完全転位と部分転位での extra half plane の動きも確認でき，それぞれの転位が原子のどのようなずれによって入るのか，より直感的見ることが可能となった．またこちらのモデルでもすべり面の上下で ab 層の並びが反転している様子を確認することができた．

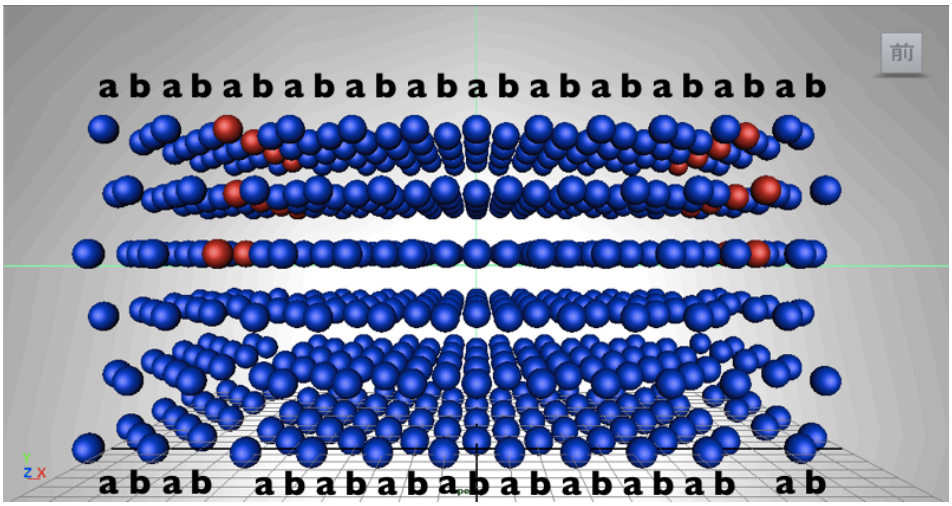


図 6.10: fcc の部分転位モデル，内部緩和前（$[1\bar{1}2]$ 方向から）．

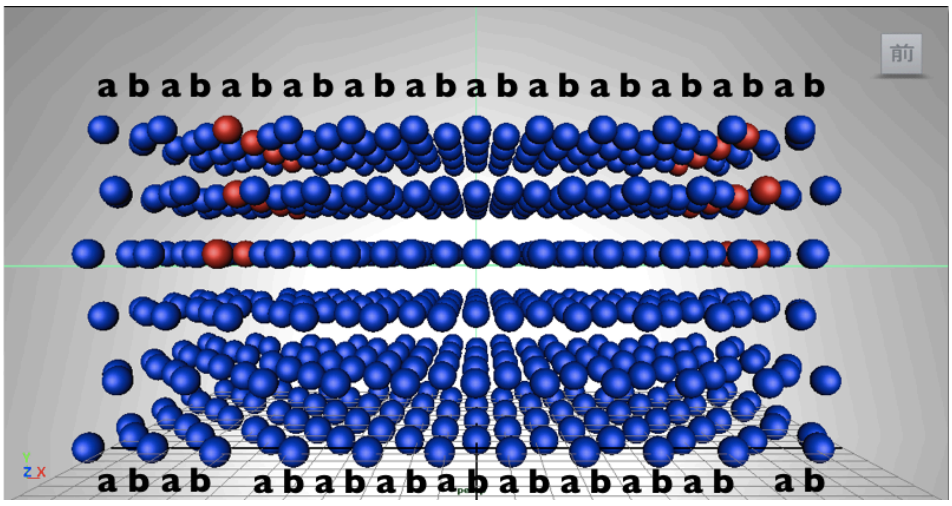


図 6.11: fcc の部分転位モデル，内部緩和後（$[1\bar{1}2]$ 方向から）．

第7章　総括

本研究の成果を以下に示す.

- 刃状転位，らせん転位の原子レベルでの様子を，3DCG で視覚化した．Maya での表示は視点を容易に変更することが可能で，転位が入った時の原子の細かな位置を見ることができる．また extra half plane やバーガース・サーキットを色を変えて表示してみることによって，対称傾角粒界で等間隔に現れる転位を，原子レベルのシミュレーションから理解するときも，視覚的観点から補うことが可能となり，研究効率が高まった.

- 静止画だけではなくアニメーションの製作も行った．完全結晶の状態から転位が起こっていく様子をアニメーションとして表現することにより，これまで文献などで載せられていた平面的な模式図とは違い，原子の動きまでもを見ることが可能となった．静止画で転位が起こる前後のモデルをただ比較するよりも，流動的で転位というものを捉え易くなったと思われる．4.4，4.6 節でアニメーションの Maya での表示，保存方法などについてもまとめてあるので，今後の他学生の研究にも役立てていただけるのではないかと思う.

- 第 6 章では，積層欠陥や部分転位について，様々な視点（方向）からの視覚化を行った．fcc 構造の原子の並びが途中で hcp 構造の並びに変化する動きや，部分転位が起こる際には $[1\bar{1}2]$ 面において，すべり面の上下で ab 列の並びが反転する理由などを絵を見ることによって，直感的に確認する事が可能となった．西谷研山本，杉本，宮本の研究での解説であるように現在では顕微鏡像などで原子の様子を見ることは可能であるが，あまりはっきりとは映っていない．その点今回の研究結果によって，さまざまな視点（方向）からの確認が可能となり，理論面での学習を行う際にはとても有効なものとなった.

引用文献

[1] C. Kittel　著,『キッテル固体物理学入門』,（丸善株式会社　2005）.

[2] 鈴木秀次　著,『転位論入門』,（株式会社アグネ　1967）.

[3] D. Hull，D. J. Bacon　著,『Introduction to Dislocations 3rd Edition』,（Pergamon Press, Oxford,　1984）.

[4] 阿部知弘　著,『MEL 教科書—Maya プログラミング入門—』,（株式会社ボーンデジタル　2004）.

[5] 戸賀瀬健介　著,『視覚化システムの構築と利用』,（関西学院大学　理工学部　情報科学科　卒業論文　2008）.

付録 A

原子座標を考える際に作成した Maple のスクリプトを以下に示す.

```
with(plottools);
restart;
with(plots): with(plottools): p:=[[0,0,0],[1,0,0],[1,1,0],[0,1,0],[0,0,1],[1,0,1],[1,1,1],[0,1,1],[-1,0,0],[-1,1,1],[1,-

points:= { seq(p[i],i=1..47) }:
pointplot3d(points,symbol=circle,symbolsize=40,color=black);

ll:=[[1,2],[2,3],[3,4],[4, 1],[1,5],[2,6],[3,7],[4, 8], [5,6],[6,7],[7,8],[8,5]]:
lines:=[seq(line(p[ll[i] [1]],p[ll[i][2]]),i=1.. nops(ll))]:
display(lines,scaling= constrained,color=black);

line1:=arrow(p[1],[1.5,1.5,0],0.05,0.3,0.1,color=blue):
line2:=arrow(p[1],[-1,1,1],0.05,0.3,0.1,color=red):
line3:=arrow(p[1],[1,-1,2],0.05,0.3,0.1,color=green):
plane1:=polygon([[1,0,-1],[0,1,0],[1,0,2],[2,-1,1]],color=blue,transparency=0.5);
plane2:=polygon([[0,-1,2],[1,0,2],[0,1,0],[-1,0,0]],color=red,transparency=0.5);
plane3:=polygon([[-1,0,0],[0,1,0],[1,0,-1],[0,-1,-1]],color=green,transparency=0.5);
plane5:=polygon([[1,0,-1],[0,-1,-1],[1,-2,1],[2,-1,1]],color=red,transparency=0.5);
plane6:=polygon([[1,0,2],[0,-1,2],[1,-2,1],[2,-1,1]],color=green,transparency=0.5);

l3:=display(lines,scaling=constrained,color=black):
p3:=pointplot3d(points, symbol=circle,symbolsize=40,color=black):
display([plane1,plane2,plane3,plane5,plane6,line1,line2,line3,p3,l3],scaling=constrained,color=black);
```

付録B

西谷研究室岩倉, 松塚の作成した内部緩和をさせるためのスクリプト, inner_relax.rb を以下に示す. 実行方法については, 4.4 節の図 4.4 を参照.

```
include Math
require 'pp'
require 'matrix'
require 'scanf'

class Atom
  attr_accessor :pos, :pos0, :nl
  def initialize(pos)
    @nl=[]
    @pos=Array.new(pos)
    @pos0=Array.new(pos)
  end
  def energy()
    ene=0
    nl.each do |j|
      ene+=ij_energy(distance(@pos,$atom_list[j].pos))
    end
    return ene
  end
  def reset_nl()
    @nl=[]
  end

  A0=0.3240268827
  E0=-1*4.0/12.0
  def ij_energy(r)#LJP
    a=1.0/(A0*r)
    return E0*((a**6 )-(a**12))
  end
end

def makeLattice()
  file = open(ARGV[0])
  atom_list=[]; n=[]; i=0
  atom_pos=[]
  while line = file.gets do
    if (res=line.chomp.scanf(" [ %f, %f, %f] "))!=[] then
      atom_pos << [res[0],res[1],res[2]]
    end
  end
  for i in 0..atom_pos.size-1 do
    atom_list << Atom.new(atom_pos[i])
  end
  file.close
  $theta = 0
  return atom_list
end

def make_L()
  $n=inputsize()
  $nx = $n[0]
  $ny = $n[1]
```

```
  $nz = $n[2]
  cf = $n[3]

  $theta=atan(1.0/cf)
  lx = 100
  ly = $ny/cos($theta)
  lz = $nz
  pp $theta
  $L=[lx,ly,lz]
end

def inputsize()
  file = open(ARGV[0])
  $n=[]
  while line = file.gets do
    if /Direct/ =~ line
    else
      n0 = line.chomp#.split(" ")
      $n << n0.to_i
    end
  end
  file.close
  return $n
end

def distance(a,b)
  tmp=0
  for i in 0..2 do
    x=a[i]-b[i]
    tmp+=x*x
  end
  return  sqrt(tmp)
end

def mkAtom_include
  atom_include=[]
  max_x = -100
  min_x = 100
  $atom_list.each do |i_a|
    if i_a.pos[0] > max_x then
      max_x = i_a.pos[0]
    elsif i_a.pos[0] < min_x then
      min_x = i_a.pos[0]
    end
  end

  $atom_list.each_with_index do |i_a,idx|
    if i_a.pos[0] < max_x-0.51*cos($theta) && i_a.pos[0] > min_x+0.51*cos($theta)
      atom_include << idx
    end
  end
  return atom_include
end

def makeNL()
  $atom_list.each do |ai| ai.reset_nl end
  nmax=$atom_list.length-1
  for i in 0..nmax do
    ai=$atom_list[i]
    for j in i+1..nmax do
      aj=$atom_list[j]
      if distance(ai.pos,aj.pos)<5.0 then
        ai.nl << j
        aj.nl << i
      end
    end
  end
```

```
end

def total_E()
  total_E=0.0
  $atom_include.each do |i|
    total_E+=$atom_list[i].energy()
  end
  return total_E
end

def diff_E(atom,j,dx)
  e0 = atom.energy()
  atom.nl.each do |i|
    if $atom_include.member?(i) then
      e0+= $atom_list[i].energy()
    end
  end
  atom.pos[j]+=dx
  e1 = atom.energy()
  atom.nl.each do |i|
    if $atom_include.member?(i) then
      e1+= $atom_list[i].energy()
    end
  end
  diff_E=e0-e1
  atom.pos[j]-=dx
  return diff_E
end

T=0.00001
$trial=0
$accept=0
def InnerRelax(trial)
  trial.times do
    makeNL()
    $atom_include.each do |i|
      atom=$atom_list[i]
      j=0
        if $MC then
          $trial+=1;$accept+=1
          dx=(1.0-2.0*rand())/10.0
          atom.pos[j]+=dx
          de=diff_E(atom,j,-dx)
          if exp(-de/T)<rand() then #mc
            $accept-=1
            atom.pos[j]-=dx
          end
        else
          $trial+=1;$accept+=1
          dx=(1.0-2.0*rand())/100.0
          atom.pos[j]+=dx
          de=diff_E(atom,j,-dx)
          if de>0 then
            $accept-=1
            atom.pos[j]-=dx
          end
        end
    end
  end
end

$atom_list=makeLattice()
$atom_include=mkAtom_include()
$t0=Time.now

$MC=true
$inner_rel=50
```

```
InnerRelax($inner_rel)

print "計算にかかった時間 ",Time.now-$t0,"秒\n"
atom_idx=[]
for i in 0..$atom_list.size-1 do
  atom_idx << i
end
if $MC
  print("mc で計算\n")
else
  print("降下法で計算\n")
end

file = File.open(ARGV[1],'w')
atom_idx.each do |i|
 file.printf("[%f,  %f,  %f]\n",$atom_list[i].pos[0],$atom_list[i].pos[1],$atom_list[i].pos[2])
end
```

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，本研究の進行に伴い，西谷研究員の皆様にも様々な知識の提供，ご協力を頂き，本研究を大成することができました．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．