結晶成長プロセスのMAYAによる視覚化

情報科学科　西谷研究室　3641 小西孝典

平成20 年2 月20 日

概要

原子レベル，連続体レベルでの結晶成長という，肉眼では到底確認できないスケールでの物理現象を CG アニメーションソフト MAYA を使用して，視覚化することを試みる．

モノが溶けるということや，凝まるということは，いったいどのようになっているのか．理科の実験等で代表されるような，飽和状態の溶液の温度を下げ，結晶が大きくなっていく実験で，どのような過程で結晶が出来上がっていくのか．その過程を今回は，Si に代わる次世代パワーデバイス材料として注目されている SiC の結晶成長を対象にして視覚化を試みる．

結晶成長の様子を，連続体レベル，原子レベルで別の見方で視覚化し，結晶成長の様子を具体的に再現することにした．

さらに，MAYA に加え，Ruby を併用することで，とくに同じものを多く作り出す場合などに効率よく作業を行うことができ，そのライブラリも付録に収録している．

目 次

第1章　緒言

原子レベルや，連続体レベルでの物質の結晶成長のように，肉眼でとらえることの出来ない物理現象を理解するには，さまざまな学習をし，そしてそれを頭でイメージして，理解をする．しかし，その学習の過程で，イメージが出来ずに学習意欲を失うことは多々あると思われる．そこで．まず視覚化したものを見ることによって，誰もが直感的にその現象を理解でき，物理現象をよりシンプルに，身近なものとしてとらえてることが出来るのではないだろうかと考える．

今回は，デバイス性能が理論限界に近づいているといわれているSiとくらべ，耐久性やオン抵抗においてすぐれ，次世代パワーデバイスとして注目されているSiCの結晶成長の視覚化を試みる．結晶成長法として，金子教授（関西学院大学理工学部）によって提唱されているMSEについてもふれている．原子レベル，連続体レベルでの視覚化をすることで，様々なスケールからのイメージが出来ると考える．

また，今回使用するMAYAで使用するMELスクリプトを，関数の宣言やコンパイルの必要がなく，誰もが自由に手に入れることが出来るRubyを用いて効率よく操作することにも試みた．

結晶成長の視覚化をするソフトも存在するが，MAYAで作ることによって，さまざまな結晶成長モデルを視覚化することができ，とても自由度が高い．

さらにRubyでライブラリ化することで，作成もとても効率がよくできるのではないかと考えられる．

第2章　パワーエレクトロニクス半導体

2.0.1　SiC

現在，最も多く用いられてるパワーエレクトロニクス半導体材料は，Siであるが，Siはデバイス性能が理論限界に近づいており，これ以上の改善が期待できない現状にある．そこで今，Siよりも耐熱性，オン抵抗に優れたSiCが注目されている．SiCは，ワイドギャップ半導体と言われ，荷電隊から伝導体に電子を持ち上げるのに必要なエネルギーがSiに比べ，約3倍必要である．このハンドギャップにより，Siに比べてSiCの絶縁破壊電解強度は10倍にもなる．これにより，同じ耐圧を得るのに，厚みが1/10にできる．またSiCはSiに比べて融点が高く，Siが200 ℃以下であるのに対し，SiCは600 ℃程度まで動作可能であり，耐熱性にも優れている．いずれもパワーでバイスに要求される特性であり，実用化が期待されている．

2.0.2　結晶成長法

従来の結晶成長法として，昇華法，Traveling Solvent法などがあるが，今回注目するのは準安定溶媒エピタキシー（MSE）である．準安定溶媒エピタキシーとは，金子教授（関西学院大学理工学部）によって提案された温度一定で駆動力を濃度差とする新たな結晶成長法である．従来の結晶成長では温度差を駆動力としてきた．ミョウバンの結晶成長がその代表例である．まず高温の水にミョウバンを高し，次にその混合水を冷却する．すると溶解度の違いから，ミョウバンの結晶が析出する．これが駆動力を温度差とする結晶成長プロセスである，しかし，準溶媒エピタキシーには温度勾配がなく，3C，4Hの科学ポテンシャルの違いから等温で基盤側と原料版側に科学ポテンシャルさが生じ，結晶成長が進む．

今回アニメーションにするのは，SiC seed に，Si の原子がくっつき，成長していく様子である．

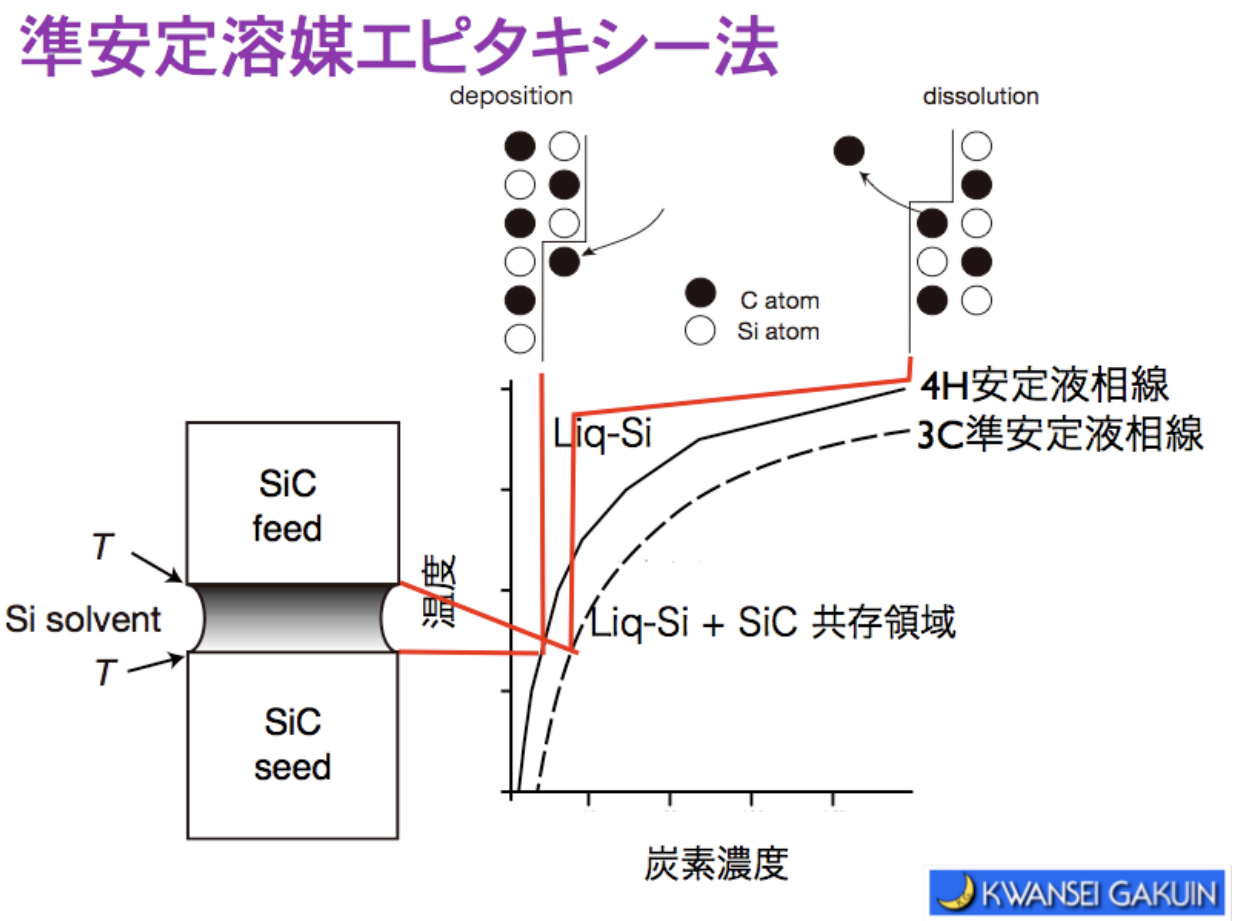

図 2.1: 準安定溶媒エピタキシーの様子.

第3章　結晶成長の様子

3.1　結晶化

3.1.1　結晶成長

生物が成長するときは細胞が大きくなって分裂を繰り返し，いわば内から育っていく．したがって，うち取り込むよう文や代謝が成長に重要な役割を果たしている．一方，結晶が大きくなるには，その表面に原子分子がくっついて太っていく．そこで，結晶の表面がどんな状態であるかが結晶成長を制御する重要な要素の一つであることが推測される．

ところで，光沢をもってキラキラ光る金属や宝石などの結晶の表面をみていると，結晶表面は真っ平らに思える．もし原子的解像度でみても平なら，この面はファセットと呼ばれる．一般に，表面では結晶原子間の結合が切れているのでエネルギーが上昇していて，これが表面エネルギーや界面エネルギーとよばれる．ファルセット面というのは原子的な結合があまり切れていなくて，エネルギーの損が少ない面である．

3.1.2　テラス，ステップ，キンク

その表面のでこぼこの説明をしていく．図2.1の表面の平らな部分をテラスという，所々単原子の段になっていて，高さの違うテラスの間の階段をステップとう．また，ステップが折れ曲がっているところをキンクとよぶ.

3.1.3　結晶化とは

そもそも，原子が結晶になる，成長する，というのはどういうことだろうか.

連続体レベルでみると，結晶成長では，まず表面に次々と原子がくっついて成長する．テラスにまず原子が吸着し，その上をまるで転がるように移動する，さらにステップにそって移動し，キンクで安定する．

これを原子レベルで見てみると，違ったイメージが現れる．まず，テラスでは，下地との結合が少ないために，原子分子が用意に表面から脱離したり，表面上の別の場所に移動したりできる．移動してステップと接触した原子は，ステップの縁の原子と結合を作れるのでエネルギー的に得である．さらにキンクまで移動すれば，さらに結合が増える．キンクまで原子分子が到達したとき，原子が結晶化したことになる．つまり，原子は完全に結晶の中に埋もれる必要はなく，キンク位置に組み込まれれば結晶化したと見なせる．表面にキンクがふんだんにあれば，結晶は成長しやすく．一方キンクが少なければ，せっかく表面に原子分子が飛び込んで来ても，中には結晶化できずに表面から飛び去るものもある．このように，成長速度は表面の荒々しさで左右される．

3.1.4 Stick & Ball

原子レベルでの表示には，Stick & Ball での表示を用いた．Stick & Ball は，原子モデルの表現に最適で，原子を Ball で表示し，原子間の結合を stick で表示．この方法は、原子の位置がわかりやすく，結合の様子がわかりやすいという両方の利点を兼ね備えた表示方法である．

第4章　アニメーション作成

4.1　使用ソフト

今回使用したソフトの特徴を以下で述べる．

4.1.1　MAYA

一口に「CG」といっても，その種類は多岐にわたっている．Adobe Photoshop や，Adobe Illustrator などのフォトレたち，ベクターデザイン系のソフトで作った絵も CG の一種であるし，工業デザインの分野で使われている，いわゆる CAD と呼ばれるものも CG に含まれるといっていい．

そんな多様性を持つ CG というジャンルにおいて，MAYA は主に映画やゲームといったエンターテイメント分野向けの CG，とりわけフォトリアリスティックな「3 DCG アニメーション」の作成を目的として，カナダのエイリアスシステムズ（現在のオートディスクと合併）が開発したソフトである．MEL スクリプトという独自の言語を用いて，MAYA を自分の作成スタイルに合わせて自在にカスタマイズできる．

MEL は，図 4.1，図 4.2，図 4.3 のように，一度実行した後に，さらに違うプログラムを実行した場合，リアルタイムに上乗せされ，同時に再現されるという特徴がある．MEL の実行したいところだけを選択（図の青くなっている部分）し，control ＋ enter コマンドで実行される．その下に，追加したい MEL を入力し，選択して実行すると，前回の実行結果はそのままに，追加される．作成したものは，自動的に名前がつくので，その物体を変化させたい場合は，名前を指定して変化させる．

３ DCG を描くというのは，絵を描くというよりもプラモデルを作る感覚に似ている，絵を描くキャンパスや漫画を書く紙は二次元としての広がりしかないが，３ DCG においては「縦」「横」のほか,「奥行き」を加えた三次元の「空間」にモノを構築していく．２ D の絵では各モチーフ

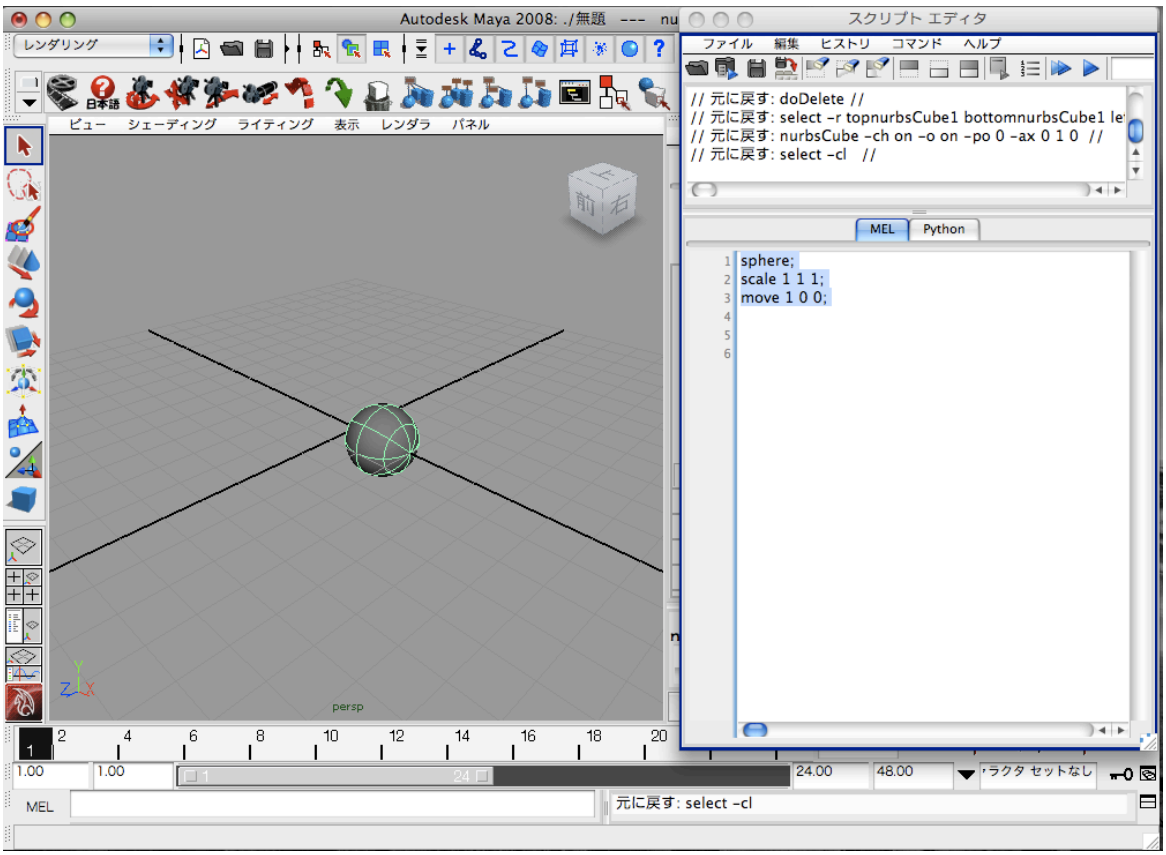

図 4.1: MEL で球体を作成.

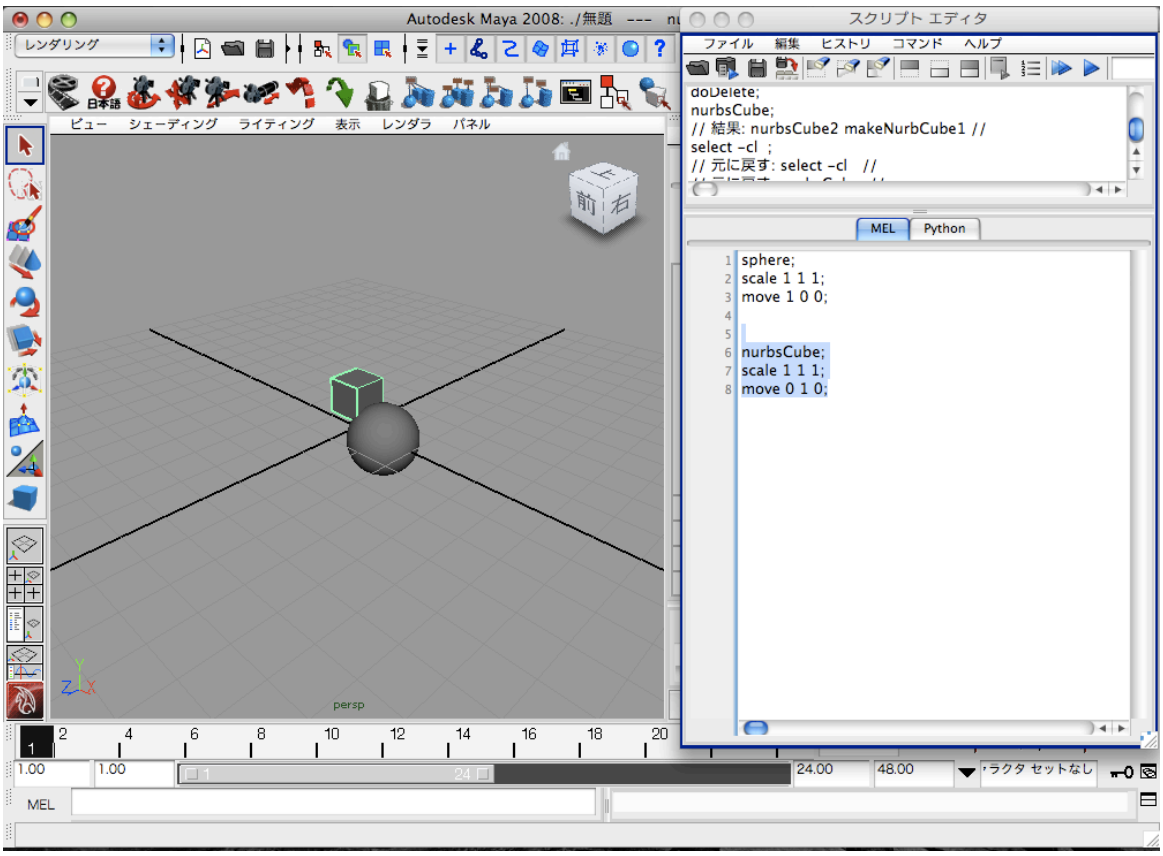

図 4.2: さらに追加で，立方体を作成.

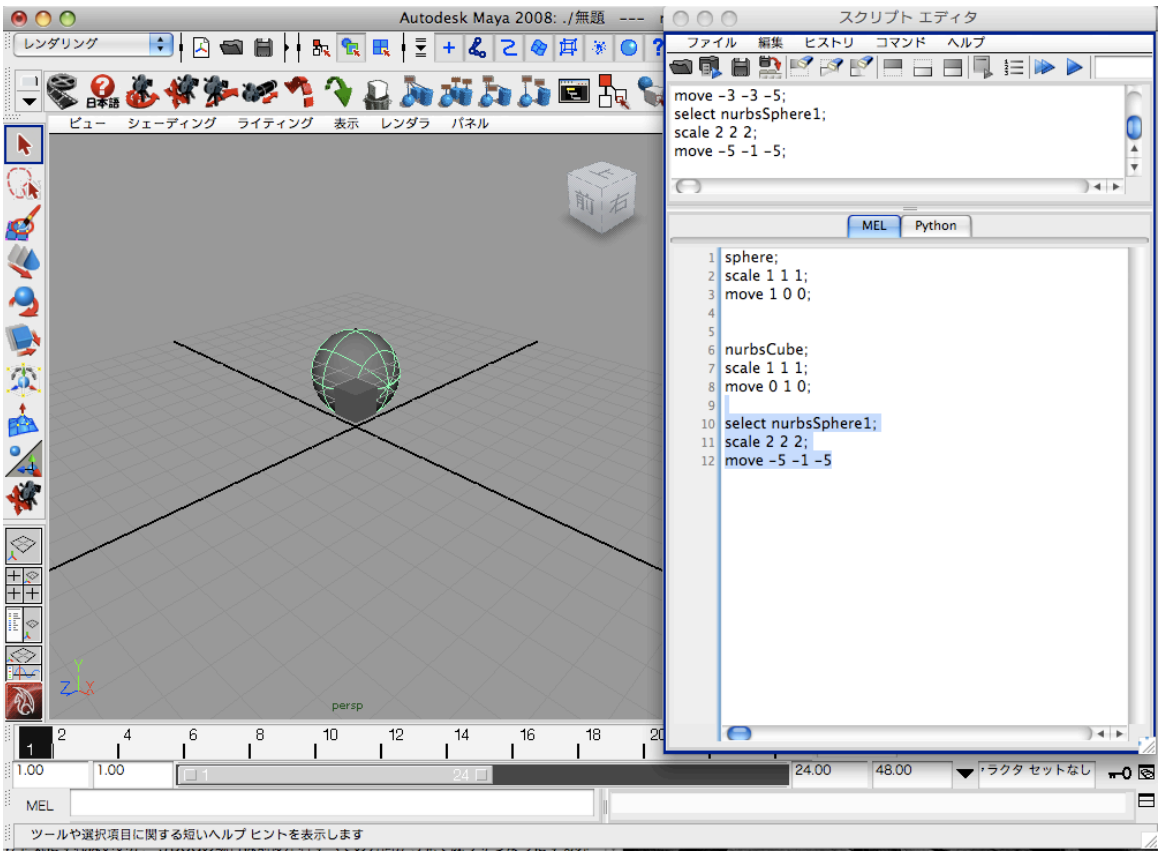

図 4.3: 名前を指定すれば，変化させることできる．

をある一方向から見たときの形だけを気にすればいいが，3 DCG の場合は前後左右すべての方向から見て成り立つようにする必要がある．

4.1.2 Ruby

Ruby はスクリプト言語である．C や Java の様なプログラム言語で書かれたプログラムを実行するには，そのプログラムのソースコードを機械命令に翻訳する「コンパイル」という作業が必要になる.「スクリプト言語」の「スクリプト」という言葉は,「プログラム」とほとんど同じような意味で使われているが，スクリプト言語の場合，書いたスクリプトはコンパイルする必要がなく，そのまま実行できる．そのため，コンパイルの必要な言語に比べ，プログラミングは手軽である．

また，Ruby はオープンソースソフトウェア（フリーソフト）である．1995 年に，インターネット上で公開され誰もが Ruby を入手して，自由に使用することが出来る．

4.2 具体的な使用例

4.2.1 Ruby から MEL への変換

本研究では Ruby，MEL 両方を使用して動画の作成をおこなった．Ruby から MEL スクリプトに書き出す手法は，同じものを多く作り出す場合には，効率よく計算できる．一方，ライトの調節などの微調整はそのあと MEL スクリプトで直接書き足すという形をとることで，両方の特徴を活用できる．

4.2.2 ライブラリの作成

X,Y,Z 軸の設定，大きさの設定など，すべてを毎回 MEL スクリプトで書いていくのは，とても面倒である．そこで，Ruby で関数をつくり，よく使う設定をつくっておき，使うときにその関数にアクセスする方法をで，アニメーション作成を行った．これによって，MEL スクリプトで毎回の用に設定をおこなう必要がなくなり，スムーズに作成ができた．

今回は，カメラの作成，色の設定，アニメーション（currentTime）の設定，証明の設定，セルの作成．オブジェクトの作成を「lib」というフォルダに収納しておき，使うときに呼び出している．

4.2.3 具体的な Ruby スクリプト

Stick & Ball での原子の様子を作成したときの，Ruby のスクリプトである．

```
include Math
require 'pp'
require "matrix"

Dir.chdir("lib"){
require 'input'
require 'unitcell'
require 'color_add'
require 'make_cell'
require 'cell_support'
require 'currentTime'
require 'object'
require 'light'
require 'camera'
}
```

ここで，lib というディレクトリに入っている，Ruby のスクリプトを関数として呼び出している．(lib の中にはいってる Ruby プログラムについては，本書の付録を参照.)

```
data_name = "4H-SiC.dat"
filename = "step_animation.mel"
```

ここで，ファイルの名前を指定.

```
unitcell,ball,p_vec,name = input(data_name)
```

ここで，データに入っている，ユニットセルのベクトル，プリミティブベクトルなどのデータの読み込みをおこなう.

```
kind = name.size
pp ball

cell_scale = [[-5,5],[0,0],[-5,10]]
for i in 0..kind-1 do
  ball[name[i]] = unitcell_basis(ball[name[i]],p_vec,cell_scale)
end
tmp = []
tmp.push ball[name[0]][0]
cell_scale = [[0,10],[1,1],[0,15]]
tmp = unitcell_basis(tmp,p_vec,cell_scale)
for i in 0..tmp.size-1 do
  ball[name[0]].push tmp[i]
end

tmp = []
tmp.push ball[name[1]][0]
cell_scale = [[0,5],[1,1],[0,10]]
tmp = unitcell_basis(tmp,p_vec,cell_scale)
for i in 0..tmp.size-1 do
  ball[name[1]].push tmp[i]
end
tmp = []
tmp.push ball[name[1]][0]
cell_scale = [[0,4],[1,1],[11,15]]
tmp = unitcell_basis(tmp,p_vec,cell_scale)
for i in 0..tmp.size-1 do
```

```
  ball[name[1]].push tmp[i]
end

tmp = []
tmp.push ball[name[0]][2]
cell_scale = [[0,5],[1,1],[0,10]]
tmp = unitcell_basis(tmp,p_vec,cell_scale)
for i in 0..tmp.size-1 do
  ball[name[0]].push tmp[i]
end
tmp = []
tmp.push ball[name[0]][2]
cell_scale = [[0,4],[1,1],[11,15]]
tmp = unitcell_basis(tmp,p_vec,cell_scale)
for i in 0..tmp.size-1 do
  ball[name[0]].push tmp[i]
end
```

キンクを作るように，必要なぶんだけセルを拡張させている．

```
################################################## MAKE_MEL
```

以下が MEL ファイルの出力データである．

```
Dir.chdir("mel"){
  file1 = File.open(filename,'w')

  color = []
  file1,color[0] = color_add(file1,[1,1,0])
  file1,color[1] = color_add(file1,[0.4,0.4,0.4])
  file1,color[2] = color_add(file1,[0.8,0.8,0.8])

  file1,color[3] = color_add(file1,[0,0,1])
  file1,color[4] = color_trans_add(file1,[0.8,0.8,0.8],[0,0,0])
  file1,color[5] = color_incan_add(file1,[1,1,1],[0.5,0.5,0.5])
```

ここで，色を決めている．

```
scale = [0.5,0.5,0.5]
for i in 0..kind-1 do
  file1 = make_ball_cell(file1,ball[name[i]],color[i],name[i],scale)
end
scale = [1,0.2,0.2]
file1 = make_stick_cell
(file1,ball[name[0]],ball[name[1]],[0,2],color[2],scale)
```

ここで，Stick と Ball を作っている．

```
file1 = object_add(file1,'sphere','background',color[5])
file1 = currentTime_move(file1,0,'background',[0,0,0])
file1 = currentTime_scale(file1,0,'background',[100,100,100])


file1 = camera_aim(file1,'camera1','camera1_aim',[25,20,30],[0,0,0])

file1.close
}
```

書き出し終了．

```
puts 'output >>> '+filename
```

4.2.4 結果の MEL スクリプト解説

結果のプログラムの一例を記す．

```
shadingNode -asShader lambert;
sets -renderable true -noSurfaceShader true -empty -name lambert1SG;
connectAttr -f lambert1.outColor lambert1SG.surfaceShader;
select -r lambert1;
setAttr "lambert1.color" -type double3 1.00000 1.00000 0.00000;
```

lambert（サーフェスを作成するマテリアルシェーダー）を作成し，
名前を付ける．
色を作成し，色にも名前をつける．
lambert の名前を指定し，指定した色をつけることができる．

```
global proc make_Si(float $x,float $y,float $z)
{
sphere;
scale 0.50000 0.50000 0.50000;
move $x $y $z;
}
global proc make_C(float $x,float $y,float $z)
{
sphere;
scale 0.50000 0.50000 0.50000;
move $x $y $z;
sets -e -forceElement lambert2SG;
}
make_Si(0.00000,0.00000,0.00000);
make_C(0.00000,1.89090,0.00000);
global proc make_cylinder1
(float $x,float $y,float $z,float $b2,float $a2)
{
cylinder;
scale 1.00000 0.20000 0.20000;
move $x $y $z;
```

```
rotate 0 $b2 $a2;
sets -e -forceElement lambert3SG;
}
make_cylinder1(0.00000,0.94545,0.00000,-0.00000,90.00000);
```

関数を作り，x,y,z 座標を指定し，作成.
これにより，同じものを多く作り出す場合，効率よく計算すること
ができる.

```
global proc make_current
(string $names[],int $c_time,float $tx,float $ty,float $tz)
{
currentTime $c_time;
setAttr ($names[0] + ".tx") $tx;
setAttr ($names[0] + ".ty") $ty;
setAttr ($names[0] + ".tz") $tz;
setKeyframe -at "tx" $names[0];
setKeyframe -at "ty" $names[0];
setKeyframe -at "tz" $names[0];
}
make_current($camera1,1,-10.00000,16.00000,0.00000);
make_current($camera1,102,30.00000,20.00000,5.00000);
make_current($camera1,120,30.00000,20.00000,5.00000);
```

これにより，currentTime でのアニメーション設定を行っている.
currentTime でのアニメーションは，指定したモノ（名前で指定）
を，指定したフレームに，指定した座標，大きさ，回転をかけて表
示し，フレーム間は，自動的に補完してくれる.

上記のプログラムは，カメラの位置を各フレームで指定している.

第5章　結果

5.1　作成したアニメーション

5.1.1　連続体レベル.

結晶の表面を連続体としてモデル化すると図5.1のような構造がある.

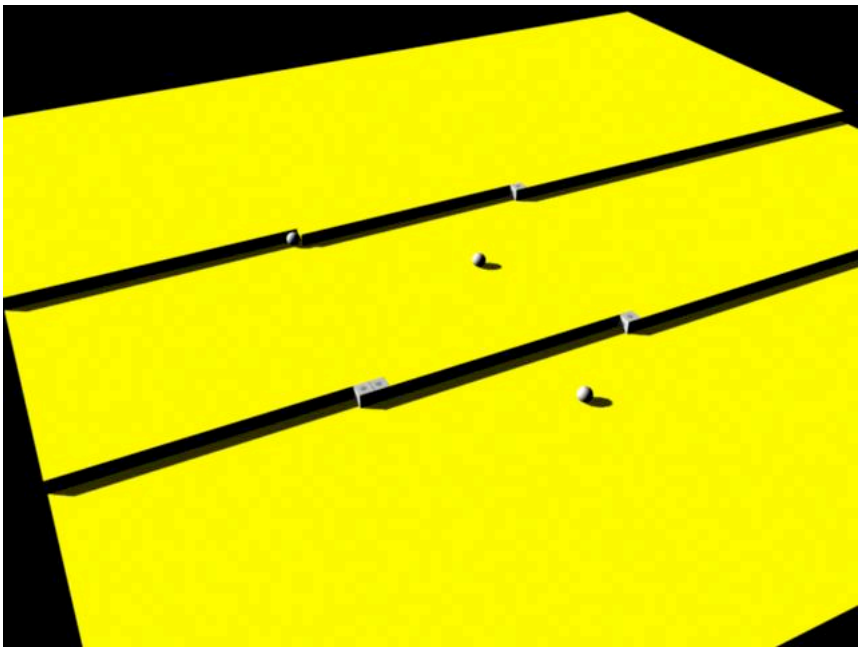

図 5.1: テラス，ステップ，キンク.

結晶が成長するときは，その表面に原子が次々とくっついて成長していく.

図5.2のようにまずテラスでは，原子が転がるように，移動する様子が再現できた.

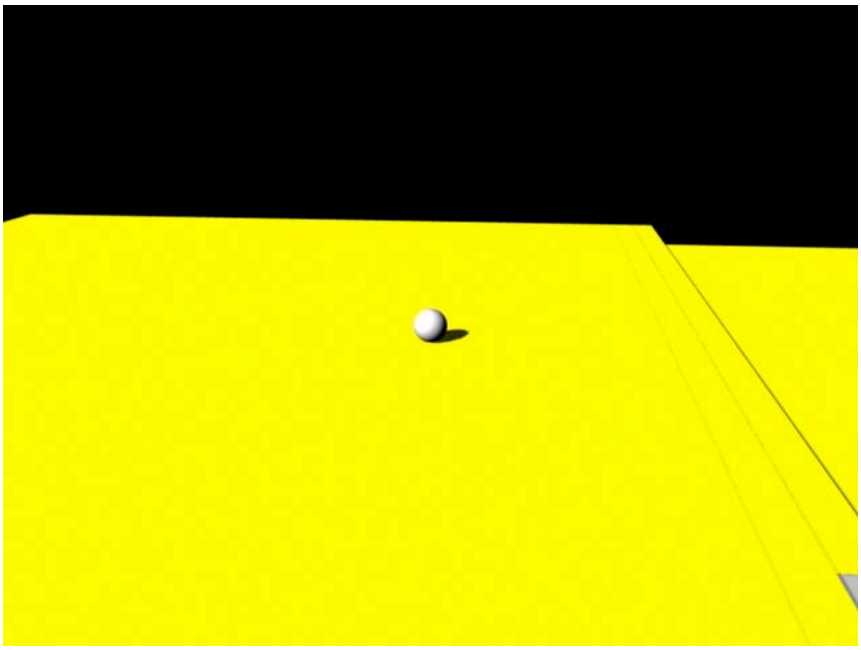

図 5.2: テラス（連続体レベル）.

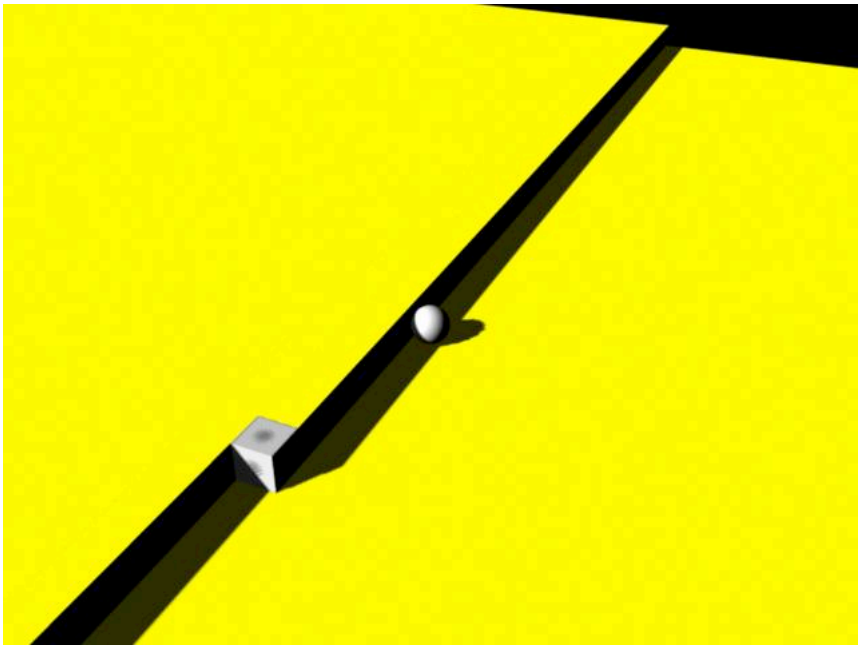

図 5.3: ステップ（連続体レベル）.

次に，図5.3のようにステップについた原子が，ステップにそって移動していく様子の再現もできた．

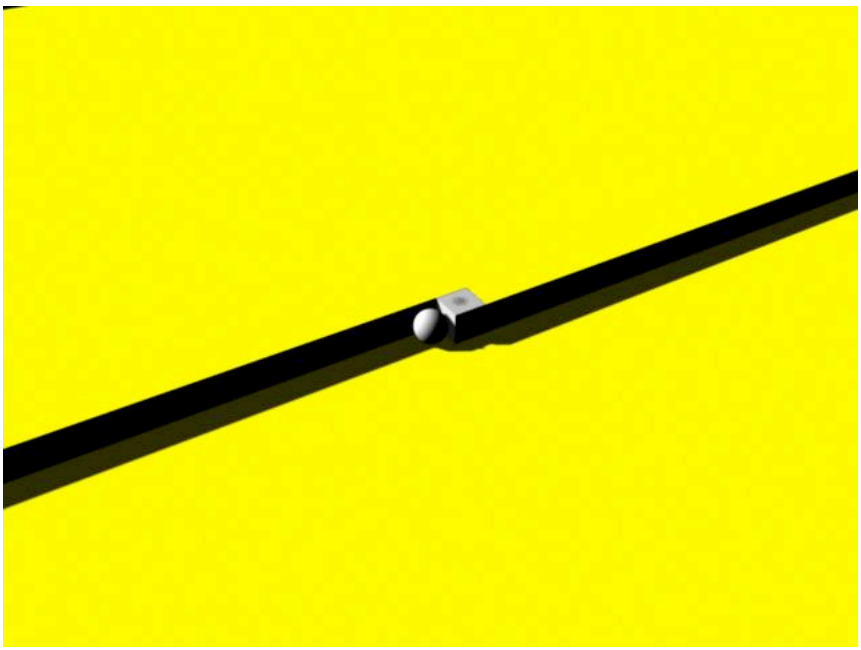

図 5.4: キンク（連続体レベル）．

最後に，図5.4のようにキンクに到達し，そこで安定している様子が再現できた．

5.1.2 原子レベル

原子レベルでのアニメーションだと，違ったイメージが出来る．

図5.5が，Rubyから MELに書き出した，Sitck & Ballの結晶の様子である．黄色のボールを Si原子とし，黒色のボールを C原子としている．

図5.6のように，テラスでは，結合が少なく，すぐ脱離してしまい，また結合を作ろうとするが，脱離しまう様子を再現できた．

図5.7のように，ステップの縁の原子と結合を作れるが，安定せずにステップにそって移動している．

図5.8キンクでもう一つ結合を作ることが出来て，ここでようやく結晶化する．

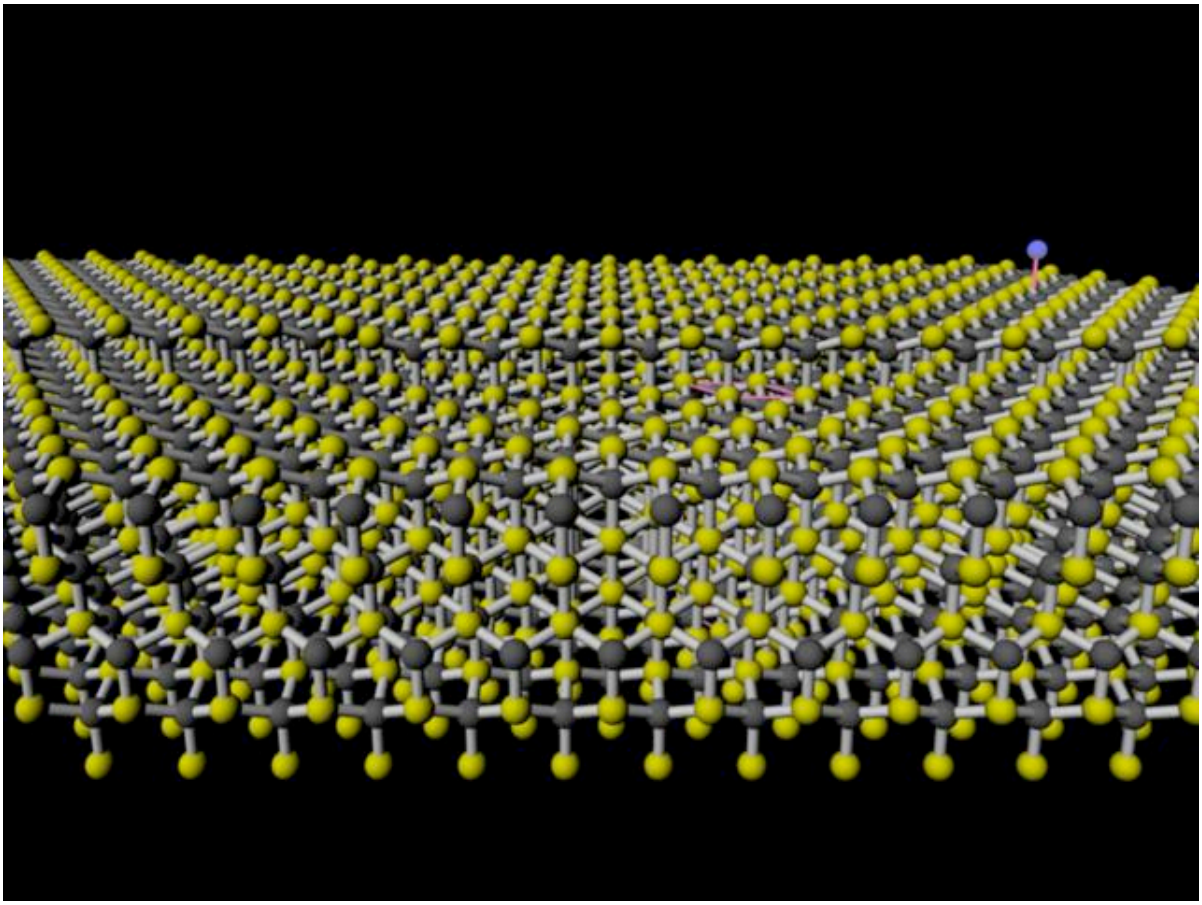

図 5.5: Ruby から MEL に書き出した SiC の Stick & Ball 模型

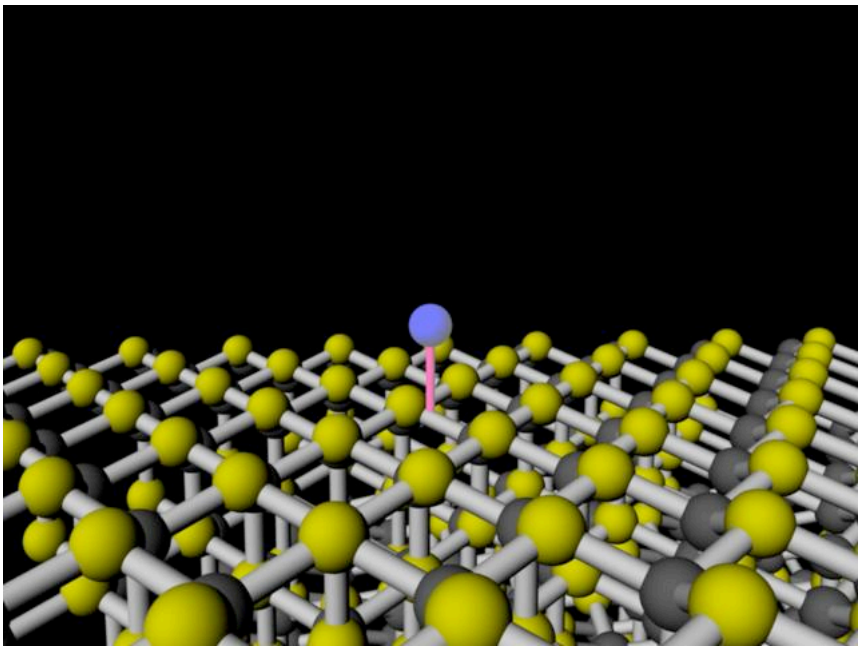

図 5.6: テラス（Stick & Ball 表示）.

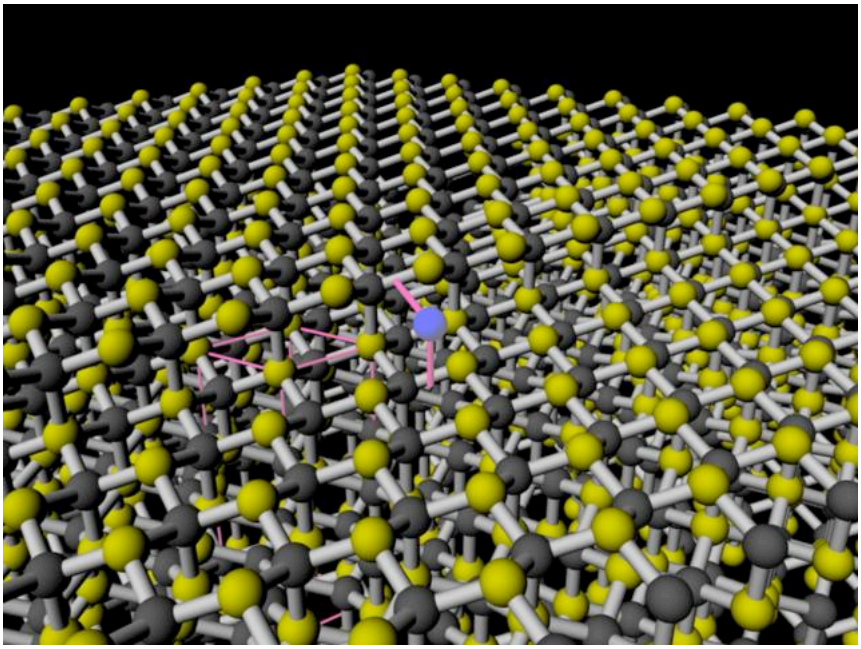

図 5.7: ステップ（Stick & Ball 表示）.

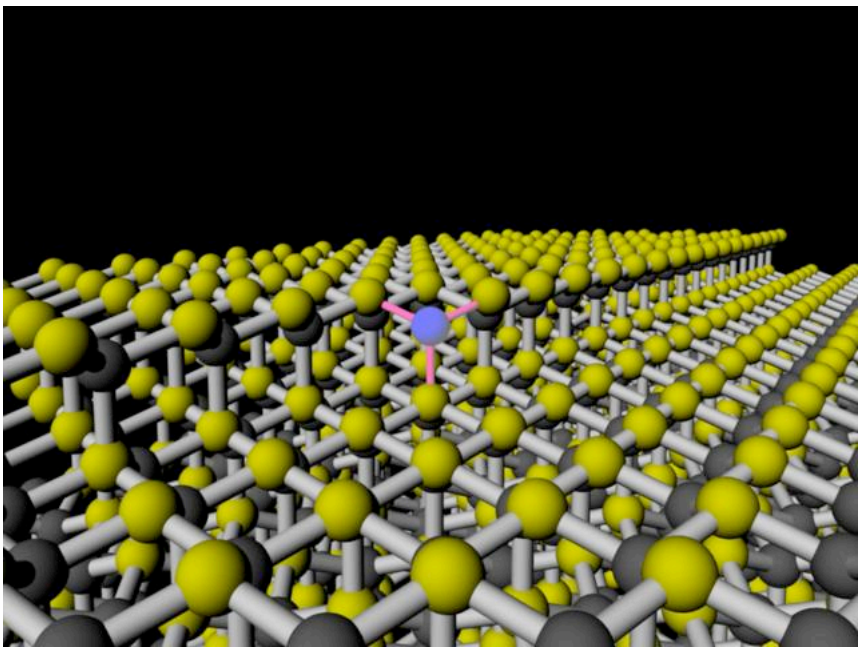

図 5.8: キンク（Stick & Ball 表示）.

第6章　総括

本研究によって，得られた知見を，ここに記す.

1. Ruby スクリプトを使って，MEL の書き出しをして，MAYA を効率よく操作することができた．これは，物理現象など，関数計算がある場合に有効である．映画やゲームなどの CG や，アニメーションなどの作成を目的とする場合ならば，MEL のみを使うよりも，MAYA で直接作品をつくり，細かい部分を MEL で書き足す方がよいとおもわれる．MEL は一度実行した後でも，プログラムを書けばリアルタイムで上乗せされるので，微調整などに使うことが多い.

2. SiC だけでなく，様々な結晶成長に応用できる．他の結晶成長視覚化ソフトと比べ．MAYA で作成した CG ならば，自由な視点から自由な構成で結晶の成長を観察できるので，視覚化としてはより理解しやすい.

3. 寺井（西谷研究室） の研究中の，VASP からの計算結果を用いれば，よりリアルな表示も可能になり，それにも対応させることが可能である．ただしそれには，基盤となる結晶の大きさの計算なども必要となってくる.

第7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，及び丁寧で的確な助言をいただ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，ご協力をいただき，本研究を成し遂げることができました．

この場を借りて，心から深くお礼申し上げます．

付 録A　ライブラリの中のプログラム

A.1　camera.rb

カメラの設定を MEL に書き出すプログラムである.

```
$camera_count = 1

def camera_aim(file1,camera,aim,xyz,aim_xyz)
  c = $camera_count
  file1.printf("camera -centerOfInterest 5 -focalLength 35
  -lensSqueezeRatio 1 -cameraScale 1
  -horizontalFilmAperture 1.4173 -horizontalFilmOffset 0
  -verticalFilmAperture 0.9449 -verticalFilmOffset 0
  -filmFit Fill -overscan 1 -motionBlur 0 -shutterAngle 144
  -nearClipPlane 0.01 -farClipPlane 1000 -orthographic 0
   -orthographicWidth 30;\n")
  file1.printf("objectMoveCommand;\n")
  file1.printf("cameraMakeNode 2 \"\";\n")
  file1.printf("setAttr \"camera%d.translateX\" %7.5f;\n",c,xyz[0])
  file1.printf("setAttr \"camera%d.translateY\" %7.5f;\n",c,xyz[1])
  file1.printf("setAttr \"camera%d.translateZ\" %7.5f;\n",c,xyz[2])
  file1.printf("setAttr \"camera
  %d_aim.translateX\" %7.5f;\n",c,aim_xyz[0])
  file1.printf("setAttr \"camera
  %d_aim.translateY\" %7.5f;\n",c,aim_xyz[1])
  file1.printf("setAttr \"camera
  %d_aim.translateZ\" %7.5f;\n",c,aim_xyz[2])
```

```
  file1.printf("select -r  camera%d;\n",c)
  file1.printf("string $%s[] = `ls -sl`;\n",camera)
  file1.printf("select -r  camera%d_aim;\n",c)
  file1.printf("string $%s[] = `ls -sl`;\n",aim)
  $camera_count += 1
  return file1
end
```

A.2 unitcell.rb

データから読み込んだユニットセルのデータを拡張するプログラムを MEL に書き出すプログラムである.

```
include Math
require 'pp'
require "matrix"

def unitcell_basis(cell,p_vec,s)
  ball = []
  for x in s[0][0]..s[0][1] do
    for y in s[1][0]..s[1][1] do
      for z in s[2][0]..s[2][1] do
        for i in 0..cell.size-1 do
          ball.push cell[i]+p_vec[0]*x+p_vec[1]*y+p_vec[2]*z
        end
      end
    end
  end
  return ball
end

def point_unitcell1
(ball1,ballE,unitcell,unitcellE,p_vec,x,y,z,i)
  if($y_vec_data=='3c')
```

```
    y_vec=Vector[0,4.36,0]
  elsif($y_vec_data=='4h')
    y_vec=Vector[0,10.08506484,0]
  elsif($y_vec_data=='6h')
    y_vec=Vector[0,15.11740000,0]
  end
  ball1.push unitcell[i]+p_vec[0]*x+p_vec[1]*z+y_vec*y
  ballE.push unitcellE[i]
  return ball1,ballE
end
def nucleus_cluster(atom1_ball,atom2_ball,p_vec)
ball1 = []
  for i in 0..atom1_ball.size-1
  ball1.push atom1_ball[i]
end
for i in 0..atom1_ball.size-1
  ball1.push atom1_ball[i]+p_vec[1]*1
end
for i in 0..atom2_ball.size-1
  ball1.push atom2_ball[i]
end
for i in 0..atom2_ball.size-1
  ball1.push atom2_ball[i]+p_vec[0]*(-1)
end
for i in 0..atom2_ball.size-1
  ball1.push atom2_ball[i]+p_vec[0]*(-1)+p_vec[2]*(-1)
end
  return ball1
end
def unitcell_nucleus(unitcell,p_vec,xyz)
  ball1 = []
  #ball1.push unitcell
  unimax = unitcell.size
  n1 = xyz[0]
  n2 = xyz[1]
```

```
  n3 = xyz[2]
  for x in -n1..n1 do
    for y in -n2..n2 do
      for z in -n3..n3 do
        for i in 0..unimax-1 do
          #if x==0 && z==0 && y==0 then
          #else
            ball1.push unitcell[i]+
            p_vec[0]*x+p_vec[1]*y+p_vec[2]*z
          #end
        end
      end
    end
  end
  return ball1
end
```

A.3 coloradd.rb

色を加えるプログラムを MEL に書き出すプログラムである.

```
require 'pp'

$color_count=1
def color_add(file1,rgb)
  cc = $color_count
  file1.printf("shadingNode -asShader lambert;\n")
  file1.printf("sets -renderable true -noSurfaceShader true
  -empty -name lambert%dSG;\n",cc)
  file1.printf("connectAttr -f lambert
  %d.outColor lambert%dSG.surfaceShader;\n",cc,cc)
  file1.printf("select -r lambert%d;\n",cc)
  file1.printf("setAttr \"lambert%d.color\" -type double3
  %1.5f %1.5f %1.5f;\n",cc,rgb[0],rgb[1],rgb[2])
```

```
  $color_count += 1
  return file1,cc
end
def color_trans_add(file1,rgb,trgb)
  cc = $color_count
  file1.printf("shadingNode -asShader lambert;\n")
  file1.printf("sets -renderable true -noSurfaceShader true
  -empty -name lambert%dSG;\n",cc)
  file1.printf("connectAttr -f lambert%d.
  outColor lambert%dSG.surfaceShader;\n",cc,cc)
  file1.printf("select -r lambert%d;\n",cc)
  file1.printf("setAttr \"lambert%d.color\" -type double3
  %1.5f %1.5f %1.5f;\n",cc,rgb[0],rgb[1],rgb[2])
  file1.printf("setAttr \"lambert%d.transparency\"
  -type double3 %1.5f %1.5f %1.5f;\n",cc,trgb[0],trgb[1],trgb[2])
  $color_count += 1
  return file1,cc
end
def color_incan_add(file1,rgb,trgb)
  cc = $color_count
  file1.printf("shadingNode -asShader lambert;\n")
  file1.printf("sets -renderable true -noSurfaceShader true
  -empty -name lambert%dSG;\n",cc)
  file1.printf("connectAttr -f lambert%d.
  outColor lambert%dSG.surfaceShader;\n",cc,cc)
  file1.printf("select -r lambert%d;\n",cc)
  file1.printf("setAttr \"lambert%d.color\" -type double3
  %1.5f %1.5f %1.5f;\n",cc,rgb[0],rgb[1],rgb[2])
  file1.printf("setAttr \"lambert%d.incandescence\"
  -type double3
  %1.5f %1.5f %1.5f;\n",cc,trgb[0],trgb[1],trgb[2])
  $color_count += 1
  return file1,cc
end
```

A.4 currentTime.rb

アニメーションの設定を MEL に書き出すプログラムである.

```
require 'pp'

$currentTime_move_count = 1
def currentTime_move(file1,time,name,xyz)
  if($currentTime_move_count==1)
    file1.printf("global proc make_current_move
    (string $names[],int $c_time,float $tx,float $ty,float $tz)\n")
    file1.printf("{\n")
    file1.printf("currentTime $c_time;\n")
    file1.printf("setAttr ($names[0] + \".tx\") $tx;\n")
    file1.printf("setAttr ($names[0] + \".ty\") $ty;\n")
    file1.printf("setAttr ($names[0] + \".tz\") $tz;\n")
    file1.printf("setKeyframe -at \"tx\" $names[0];\n")
    file1.printf("setKeyframe -at \"ty\" $names[0];\n")
    file1.printf("setKeyframe -at \"tz\" $names[0];\n")
    file1.printf("}\n")
  end
  file1.printf("make_current_move
  ($%s,%d,%7.5f,%7.5f,%7.5f);
  \n",name,time,xyz[0],xyz[1],xyz[2])
  $currentTime_move_count += 1
  return file1
end

$currentTime_rotate_count = 1
def currentTime_rotate(file1,time,name,xyz)
  if($currentTime_rotate_count==1)
    file1.printf("global proc make_current_rotate
    (string $names[],int $c_time,float $tx,float $ty,float $tz)\n")
    file1.printf("{\n")
    file1.printf("currentTime $c_time;\n")
    file1.printf("setAttr ($names[0] + \".rx\") $tx;\n")
```

```
    file1.printf("setAttr ($names[0] + \".ry\") $ty;\n")
    file1.printf("setAttr ($names[0] + \".rz\") $tz;\n")
    file1.printf("setKeyframe -at \"rx\" $names[0];\n")
    file1.printf("setKeyframe -at \"ry\" $names[0];\n")
    file1.printf("setKeyframe -at \"rz\" $names[0];\n")
    file1.printf("}\n")
  end
  file1.printf("make_current_rotate
  ($%s,%d,%7.5f,%7.5f,%7.5f);\n",
  name,time,xyz[0],xyz[1],xyz[2])
  $currentTime_rotate_count += 1
  return file1
end

$currentTime_scale_count = 1
def currentTime_scale(file1,time,name,xyz)
  if($currentTime_scale_count==1)
    file1.printf("global proc make_current_scale
    (string $names[],int $c_time,float $tx,float $ty,float $tz)\n")
    file1.printf("{\n")
    file1.printf("currentTime $c_time;\n")
    file1.printf("setAttr ($names[0] + \".sx\") $tx;\n")
    file1.printf("setAttr ($names[0] + \".sy\") $ty;\n")
    file1.printf("setAttr ($names[0] + \".sz\") $tz;\n")
    file1.printf("setKeyframe -at \"sx\" $names[0];\n")
    file1.printf("setKeyframe -at \"sy\" $names[0];\n")
    file1.printf("setKeyframe -at \"sz\" $names[0];\n")
    file1.printf("}\n")
  end
  file1.printf("make_current_scale
  ($%s,%d,%7.5f,%7.5f,%7.5f);\n"
  ,name,time,xyz[0],xyz[1],xyz[2])
  $currentTime_scale_count += 1
  return file1
end
```

```
def currentTime_color(file1,time,color,xyz)
  file1.printf("currentTime %d;\n",time)
  file1.printf("select -r lambert%d;\n",color)
  file1.printf("setAttr \"lambert%d.color\" -type double3 ",color)
  file1.printf(" %7.5f %7.5f %7.5f;\n",xyz[0],xyz[1],xyz[2])
  file1.printf("setKeyframe -breakdown 0
   -hierarchy none -controlPoints 0 -shape 0
    {\"lambert%d\"};\n",color)
  return file1
end
def currentTime_color_trans(file1,time,color,xyz)
  file1.printf("currentTime %d;\n",time)
  file1.printf("select -r lambert%d;\n",color)
  file1.printf("setAttr \"lambert%d.transparency\
  " -type double3 ",color)
  file1.printf(" %7.5f %7.5f %7.5f;\n",xyz[0],xyz[1],xyz[2])
  file1.printf("setKeyframe -breakdown 0
   -hierarchy none -controlPoints 0 -shape 0
    {\"lambert%d\"};\n",color)
  return file1
end
```

A.5 light.rb

照明の設定を MEL に書き出すプログラムである.

```
$light_count = 1

def ambientLight(file1,tmp,xyz,rxyz,sxyz)
  c = $light_count
  file1.printf("defaultAmbientLight(1,1,1,1,\"0\",0,0,0,0);\n")
  file1.printf("setAttr \"ambientLight
  %d.translateX\" %7.5f;\n",c,xyz[0])
```

```
  file1.printf("setAttr \"ambientLight
  %d.translateY\" %7.5f;\n",c,xyz[1])
  file1.printf("setAttr \"ambientLight
  %d.translateZ\" %7.5f;\n",c,xyz[2])
  file1.printf("setAttr \"ambientLight
  %d.rotateX\" %7.5f;\n",c,rxyz[0])
  file1.printf("setAttr \"ambientLight
  %d.rotateY\" %7.5f;\n",c,rxyz[1])
  file1.printf("setAttr \"ambientLight
  %d.rotateZ\" %7.5f;\n",c,rxyz[2])
  file1.printf("setAttr \"ambientLight
  %d.scaleX\" %7.5f;\n",c,sxyz[0])
  file1.printf("setAttr \"ambientLight
  %d.scaleY\" %7.5f;\n",c,sxyz[1])
  file1.printf("setAttr \"ambientLight
  %d.scaleZ\" %7.5f;\n",c,sxyz[2])
  file1.printf("setAttr \"ambientLightShape
  %d.intensity\" %1.5f;\n",c,tmp)
  $light_count += 1
  return file1
end
def derectionalLight(file1,tmp,x,y,z,rx,ry,rz,sx,sy,sz)
  c = $light_count
  file1.printf("defaultDirectionalLight
  (1,1,1,1,\"0\",0,0,0,0);\n")
  file1.printf("setAttr \"directionalLight
  %d.translateX\" %7.5f;\n",c,xyz[0])
  file1.printf("setAttr \"directionalLight
  %d.translateY\" %7.5f;\n",c,xyz[1])
  file1.printf("setAttr \"directionalLight
  %d.translateZ\" %7.5f;\n",c,xyz[2])
  file1.printf("setAttr \"directionalLight
  %d.rotateX\" %7.5f;\n",c,rxyz[0])
  file1.printf("setAttr \"directionalLight
  %d.rotateY\" %7.5f;\n",c,rxyz[1])
```

```
  file1.printf("setAttr \"directionalLight
  %d.rotateZ\" %7.5f;\n",c,rxyz[2])
  file1.printf("setAttr \"directionalLight
  %d.scaleX\" %7.5f;",c,sxyz[0])
  file1.printf("setAttr \"directionalLight
  %d.scaleY\" %7.5f;",c,sxyz[1])
  file1.printf("setAttr \"directionalLight
  %d.scaleZ\" %7.5f;",c,sxyz[2])
  file1.printf("setAttr \"directionalLightShape
  %d.intensity\" %1.5f;\n",c,tmp)
  $light_count += 1
  return file1
end
def pointLight(file1,tmp,x,y,z,rx,ry,rz,sx,sy,sz)
  c = $light_count
  file1.printf("defaultPointLight
  (1,1,1,1,\"0\",0,0,0,0);\n")
  file1.printf("setAttr \"pointLight
  %d.translateX\" %7.5f;\n",c,x)
  file1.printf("setAttr \"pointLight
  %d.translateY\" %7.5f;\n",c,y)
  file1.printf("setAttr \"pointLight
  %d.translateZ\" %7.5f;\n",c,z)
  file1.printf("setAttr \"pointLight
  %d.rotateX\" %7.5f;\n",c,rx)
  file1.printf("setAttr \"pointLight
  %d.rotateY\" %7.5f;\n",c,ry)
  file1.printf("setAttr \"pointLight
  %d.rotateZ\" %7.5f;\n",c,rz)
  file1.printf("setAttr \"pointLight
  %d.scaleX\" %7.5f;\n",c,sx)
  file1.printf("setAttr \"pointLight
  %d.scaleY\" %7.5f;\n",c,sy)
  file1.printf("setAttr \"pointLight
  %d.scaleZ\" %7.5f;\n",c,sz)
```

```
  file1.printf("setAttr \"pointLightShape
  %d.intensity\" %1.5f;\n",c,tmp)
  $light_count += 1
  return file1
end
def spotLight(file1,tmp,x,y,z,rx,ry,rz,sx,sy,sz)
  c = $light_count
  file1.printf("defaultSpotLight(1,1,1,1,\"0\",0,0,0,0);\n")
  file1.printf("setAttr \"spotLight
  %d.translateX\" %7.5f;\n",c,xyz[0])
  file1.printf("setAttr \"spotLight
  %d.translateY\" %7.5f;\n",c,xyz[1])
  file1.printf("setAttr \"spotLight
  %d.translateZ\" %7.5f;\n",c,xyz[2])
  file1.printf("setAttr \"spotLight
  %d.rotateX\" %7.5f;\n",c,rxyz[0])
  file1.printf("setAttr \"spotLight
  %d.rotateY\" %7.5f;\n",c,rxyz[1])
  file1.printf("setAttr \"spotLight
  %d.rotateZ\" %7.5f;\n",c,rxyz[2])
  file1.printf("setAttr \"spotLight
  %d.scaleX\" %7.5f;\n",c,sxyz[0])
  file1.printf("setAttr \"spotLight
  %d.scaleY\" %7.5f;\n",c,sxyz[1])
  file1.printf("setAttr \"spotLight
  %d.scaleZ\" %7.5f;\n",c,sxyz[2])
  file1.printf("setAttr \"spotLightShape
  %d.intensity\" %1.5f;\n",c,tmp)
  $light_count += 1
  return file1
end
def areaLight(file1,tmp,x,y,z,rx,ry,rz,sx,sy,sz)
  c = $light_count
  file1.printf("defaultAreaLight
  (1,1,1,1,\"0\",0,0,0,0);\n")
```

```
  file1.printf("setAttr \"areaLigh
  t%d.translateX\" %7.5f;\n",c,xyz[0])
  file1.printf("setAttr \"areaLigh
  t%d.translateY\" %7.5f;\n",c,xyz[1])
  file1.printf("setAttr \"areaLigh
  t%d.translateZ\" %7.5f;\n",c,xyz[2])
  file1.printf("setAttr \"areaLigh
  t%d.rotateX\" %7.5f;\n",c,rxyz[0])
  file1.printf("setAttr \"areaLigh
  t%d.rotateY\" %7.5f;\n",c,rxyz[1])
  file1.printf("setAttr \"areaLigh
  t%d.rotateZ\" %7.5f;\n",c,rxyz[2])
  file1.printf("setAttr \"areaLigh
  t%d.scaleX\" %7.5f;\n",c,sxyz[0])
  file1.printf("setAttr \"areaLigh
  t%d.scaleY\" %7.5f;\n",c,sxyz[1])
  file1.printf("setAttr \"areaLigh
  t%d.scaleZ\" %7.5f;\n",c,sxyz[2])
  file1.printf("setAttr \"areaLightShape
  %d.intensity\" %1.5f;\n",c,tmp)
  $light_count += 1
  return file1
end
def volumeLight(file1,tmp,x,y,z,rx,ry,rz,sx,sy,sz)
  c = $light_count
  file1.printf("defaultVolumeLight
  (1,1,1,1,\"0\",0,0,0,0);\n")
  file1.printf("setAttr \"volumeLigh
  t%d.translateX\" %7.5f;\n",c,xyz[0])
  file1.printf("setAttr \"volumeLigh
  t%d.translateY\" %7.5f;\n",c,xyz[1])
  file1.printf("setAttr \"volumeLigh
  t%d.translateZ\" %7.5f;\n",c,xyz[2])
  file1.printf("setAttr \"volumeLight
  %d.rotateX\" %7.5f;\n",c,rxyz[0])
```

```
  file1.printf("setAttr \"volumeLight
  %d.rotateY\" %7.5f;\n",c,rxyz[1])
  file1.printf("setAttr \"volumeLight
  %d.rotateZ\" %7.5f;\n",c,rxyz[2])
  file1.printf("setAttr \"volumeLight
  %d.scaleX\" %7.5f;\n",c,sxyz[0])
  file1.printf("setAttr \"volumeLight
  %d.scaleY\" %7.5f;\n",c,sxyz[1])
  file1.printf("setAttr \"volumeLight
  %d.scaleZ\" %7.5f;\n",c,sxyz[2])
  file1.printf("setAttr \"volumeLightShape
  %d.intensity\" %1.5f;\n",c,tmp)
  $light_count += 1
  return file1
end
```

A.6 makecell.rb

セルを作るプログラムを MEL に書き出すプログラムである.

```
include Math

$cell_ball_count = 0
$cell_cylinder_count = 0
$cylinder_proc_count = 0

def make_sphere_proc(file1,color,name,scale)
  file1.printf("global proc make_%s
  (float $x,float $y,float $z)\n",name)
  file1.printf("{\n")
  file1.printf("sphere;\n")
  file1.printf("scale %7.5f %7.5f %7.5f;\n",scale[0],scale[1],scale[2])
  file1.printf("move $x $y $z;\n")
  if(color!=1)
```

```
    file1.printf("sets -e -forceElement lambert%dSG;\n",color)
  end
  file1.printf("}\n")
  return file1
end
def make_ball_cell(file1,ball,color,name,scale)
  file1 = make_sphere_proc(file1,color,name,scale)
  for i in 0..ball.size-1 do
    file1.printf("make_%s
    (%7.5f,%7.5f,%7.5f);\n",name,ball[i][0],ball[i][1],ball[i][2])
  end
  return file1
end

def make_cylinder_proc(file1,color,scale)
  $cylinder_proc_count += 1
  d = $cylinder_proc_count
  file1.printf("global proc make_cylinder%d
  (float $x,float $y,float $z,float $b2,float $a2)\n",d)
  file1.printf("{\n")
  file1.printf("cylinder;\n")
  file1.printf("scale %7.5f %7.5f %7.5f;\n",
  scale[0],scale[1],scale[2])
  file1.printf("move $x $y $z;\n")
  file1.printf("rotate 0 $b2 $a2;\n")
  file1.printf("sets -e -forceElement lambert%dSG;\n",color)
  file1.printf("}\n")
  return file1
end
def make_cylinder(file1,ball1,ball2,comb_tmp)
  comb_tmp.each{|ij|
    half_v = (ball1[ij[0]]+ball2[ij[1]])*0.5
    tmp = ball1[ij[0]]-ball2[ij[1]]
    if tmp[1] < 0  then
      dir_vec=ball2[ij[1]]-ball1[ij[0]]
```

```
    else
      dir_vec=ball1[ij[0]]-ball2[ij[1]]
    end
    x=dir_vec[0]
    y=dir_vec[1]
    z=dir_vec[2]
    aa=1/sqrt(x**2+y**2+z**2)
    x1=x*aa
    y1=y*aa
    z1=z*aa
    b1=-asin(z1)
    a1=acos(x1/cos(b1))
    a2=a1*180/PI
    b2=b1*180/PI
    file1.printf("make_cylinder%d
    (%7.5f,%7.5f,%7.5f,%7.5f,%7.5f);\n",
    $cylinder_proc_count,half_v[0],half_v[1]
    ,half_v[2],b2,a2)
    $cell_cylinder_count += 1
  }
  return file1
end
def make_stick_cell(file1,ball1,ball2,len,color,scale)
  n1 = ball1.size
  n2 = ball2.size
  comb_tmp=[]

  for i in 0..n1-1 do
    for j in 0..n2-1 do
      tmp_vector=ball1[i]-ball2[j]
      l = tmp_vector.r
      if(l!=0)
        flag = 0
        if(len[0]<=l && l<len[1])
          if(comb_tmp.size>0)
```

```
            comb_tmp.each do |a|
              if(a[0]==j && a[1]==i)
                flag = 1
                break
              end
            end
          end
          if(flag==0)
            comb_tmp.push([i,j])
          end
        end
      end
    end
  end

  file1 = make_cylinder_proc(file1,color,scale)
  file1 = make_cylinder(file1,ball1,ball2,comb_tmp)
  return file1
end
def cylinder_cell(ball1,ball2,len)
  n1 = ball1.size
  n2 = ball2.size
  comb_tmp=[]

  for i in 0..n1-1 do
    for j in 0..n2-1 do
      tmp_vector=ball1[i]-ball2[j]
      l = tmp_vector.r
      if(l!=0)
        flag = 0
        if(len[0]<=l && l<len[1])
          if(comb_tmp.size>0)
            comb_tmp.each do |a|
              if(a[0]==j && a[1]==i)
                flag = 1
```

```
                break
              end
            end
          end
          if(flag==0)
            comb_tmp.push([i,j])
          end
        end
      end
    end
  end
  cell_cylinder_array = []
  comb_tmp.each{|ij|
    half_v = (ball1[ij[0]]+ball2[ij[1]])*0.5
    tmp = ball1[ij[0]]-ball2[ij[1]]
    if tmp[1] < 0  then
      dir_vec=ball2[ij[1]]-ball1[ij[0]]
    else
      dir_vec=ball1[ij[0]]-ball2[ij[1]]
    end
    x=dir_vec[0]
    y=dir_vec[1]
    z=dir_vec[2]
    aa=1/sqrt(x**2+y**2+z**2)
    x1=x*aa
    y1=y*aa
    z1=z*aa
    b1=-asin(z1)
    a1=acos(x1/cos(b1))
    a2=a1*180/PI
    b2=b1*180/PI
    cell_cylinder_array.push [half_v[0],half_v[1],half_v[2]]
  }
  return cell_cylinder_array
end
```

関連図書

[1] 西主森武, 「AUTODESK MAYA　オフィシャルトレーニングブック」　（ワークスコーポレーション　2006）.

[2] 斉藤幸夫, 「結晶成長」（裳華房フィジックスライブラリー　2002）.

[3] Chris Pine, 「初めてのプログラミング」（O'REILLY 2006）.

[4] 高橋征義　後藤裕蔵, 「たのしいRuby」（ソフトバンククリエイティブ株式会社　2002）.

[5] 坂本憲　関西学院大学修士論文　「SiC単結晶成長の原子レベルシミュレーション」（2008）.