MAYAによるSiC結晶構造の視覚化

理工学部　西谷研究室　4703　鍋島　麻希

平成20年2月21日

概要

高校において物理を履修していない学生にとって，大学でおこなっている物理計算は何をしているのかを理解するだけでも苦労する．物理現象を視覚化することで，容易に理解できるようにすることが本研究の目的である．
これには，3D アニメーション作成ソフトウェア MAYA[1] を用いて視覚化を行った．MAYA には独自のスクリプト言語である MEL[2] がある．この MEL スクリプトを自動生成するスクリプト言語 Ruby[3] で作成した．作成対象としては，原子レベルでの結晶成長や，第一原理計算で提供されるエネルギー計算を補うために，結晶構造，結晶表面，結晶断面を視覚化した．本研究で対象とする結晶は SiC で，結晶構造は SiC の 3C,4H,6H の 3 構造を，結晶表面は典型的な面である (0001) 面,(1-100) 面,(11-20) 面である．

目 次

第1章　序論

高校や大学で物理を履修していない学生にとって，大学で行っている物理計算は何をしているのかを理解するだけでも苦労する．物理学とは，自然界に見られる現象の法則を捉えるもので，人が直接見ることが出来ないものを理解するのは難しい．物理現象の一例として，図1.1はSiCの結晶構造の積層周期の違いを示したものである．物理を専門としていない学生にとって，図1.1が何を意図しているのかを理解するのは困難である．

そこで，物理現象を容易に理解出来る有効な手法として視覚化があげられる．視覚化とは，人間が直接見ることのできないデータを，見ることのできるもの (グラフ，図，表など) にする手法の総称である．今回，視覚化の方法としてMAYAを用いる．MAYAは数ある3Dモデリングソフトの中でも,MELをスクリプトとして，利用出来，よりサイエンティフィックに記述することが出来る．

今回の研究では，MAYAを利用して，図1.1のような結晶構造，結晶断面の視覚化を試みた．MAYAでは，切断したいところに面を挿入し，その面の透明度を上げることによって，切断した面から見た時に立体的に捉えることを容易にする．それによって，切断したところのballとstickが浮き上がって見え，どのような構造をとっているか理解しやすいようにする．

1.1　3C,4H,6H-SiC結晶構造

SiCにはその積層周期の違いから200種類もの結晶多形が存在する．その中でも主なものは六方晶の4H,6H及び立方晶の3Cの３種類である．これら結晶多形の表記方法は，積層方向の単位格子に含まれるSi-Cの単位層の数と，結晶系の頭文字 (C:立方晶，H:六方晶) を組み合わせて表されている．4Hは主にパワーデバイス用として，6Hは主に青色発行素子な

どの GaN 用基板として使われている．SiC は Si よりも省エネ効果が得られるため，今後実用化が期待されている．その SiC の新たな結晶成長として MSE が，関西学院大学の金子教授らによって提案された．これは，3C,4H-SiC の化学ポテンシャルの違いから結晶成長が進むものである．このように, 単結晶成長プロセスである MSE は,SiC の 3C,4H,6H という結晶構造の違いが支配しているため，その詳しい理解が研究の出発点となる．図 1.1 に 3C,4H,6H の積層周期を示した．

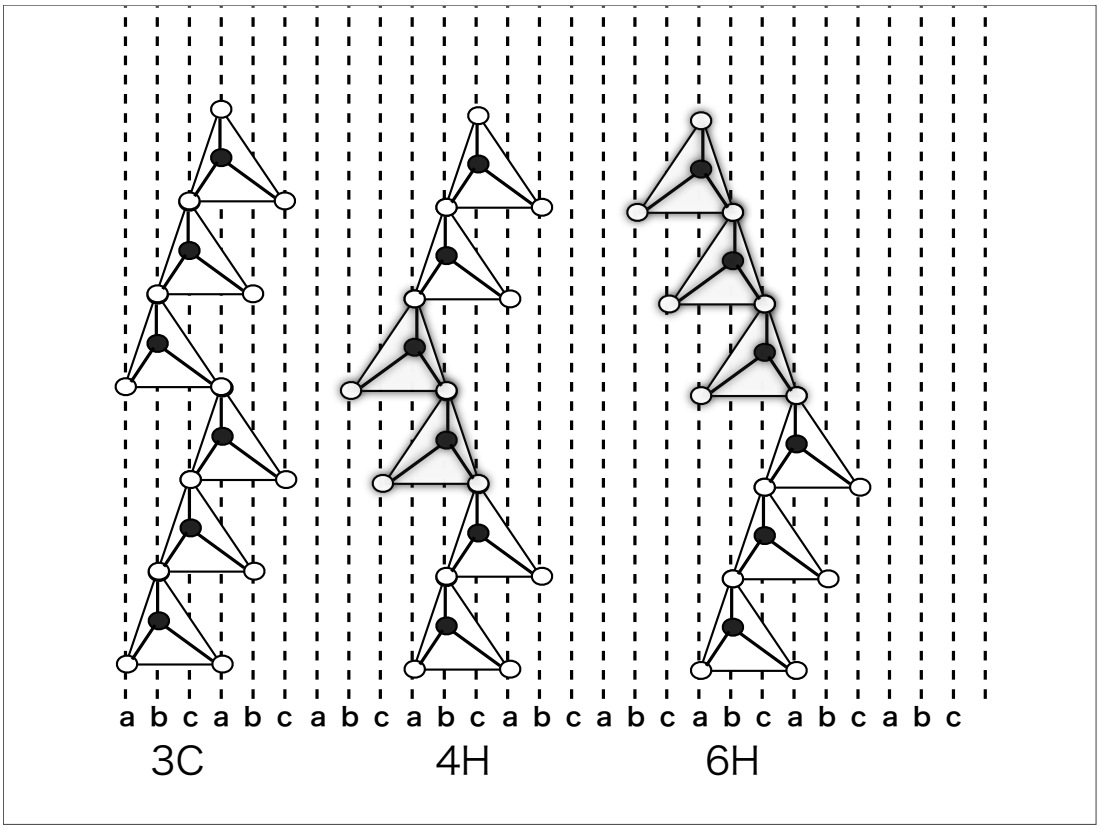

図 1.1: 3c,4h,6h の結晶構造 ((11-20) 面から見た) 模式図.

1.2 MAYA

結晶モデルを構築するソフトは多くあり，西谷研究室でも MedeA を利用してきたが，切断面などを詳しく視覚することが出来ない．そこで，

MAYA を用いることにした．MAYA は，オープンアーキテクチャを基盤とした強力な統合型 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 MAYA を使用しており，自動車メーカーなどの製造業では，デザイン的な要素の強い部品の設計に使用している [1].
このように，MAYA はグラフィックス性に優れており，自由度も高い．また，アニメーションの作成も可能なので，難解な物理現象も容易に理解することが出来る．また，西谷研究室の寺井が表面エネルギーの第一原理計算の研究をしており，SiC の結晶構造，結晶表面の違いを理解することが必要となった．

第2章　手法

2.1　RubyとMAYAによる格子モデルのstick&ball法による表示

2.1.1　スクリプト言語

MAYA はその全て動作をスクリプト言語である MEL (Maya Embedded Language) [2] で記述することが可能である．しかし，MEL はグラフィックスに特化しており，一般的な言語に比べてループや線形代数計算の記述が複雑になる．そこで，よりシンプルなスクリプト言語である Ruby[3] を用いて，MEL を出力するコードを作成した．Ruby はオブジェクト指向スクリプト言語である．オブジェクト指向とは，プログラムの操作手順よりも操作の対象に重点を置く考えで，プログラムを書いて実行するまでの時間が比較的短く済む．Ruby は変数に型を持たない動的片付け，コンパイル不要で書いてすぐ動かせるインタプリタ方式の仕組みを持つ．また，スクリプトエディタに Emacs を使用出来，Emacs は MEL のスクリプトエディタと違い，高機能でフォントが色分けされるなど，カスタマイズ性の高いスクリーンエディタである． よって，これらの特徴から Ruby で記述することによって，より効率的にプログラムを書くことが出来た．

2.1.2　stick&ball法

原子レベルの視覚化に適した stick&ball 表示法を用いた．これは，原子を ball(球) で，原子間の結合を stick(棒) で表わす手法である．対象とする結晶構造のユニットセルから，結晶格子をスーパーセルに拡げ ball の位置を計算する．また，簡単な線形代数計算によって，2 つの原子位置から stick の中心位置，角度を計算する．さらに結晶断面の挿入や切り取り，結晶表面を見れるようにした．

2.2　3次元変換のマトリックス表現

MAYA は 3 次元コンピュータグラフィックスソフトウェアである．そのため，原子間の結合の stick の中心位置，角度を決定するのに 3 次元変換の 3 × 3 のマトリックスで表すことが出来る．本研究では，3 次元表現に，数学での標準的な慣習である右手座標系を用いる．慣習により，右手座標系における，正の回転とは 1 つの正軸から原点に向かって見ながらもう 1 つの正軸を反時計周りに 90 °の回転させたときに，残りの軸に一致する回転のことをいう．この慣習を, 表 3.5 に示す．これを図に表したものが，図 2.1 である.[4]

表 2.1: 3 次元表現の右手座標系の慣習.

回転軸	正の回転方向
x	y から z
y	z から x
z	x から y

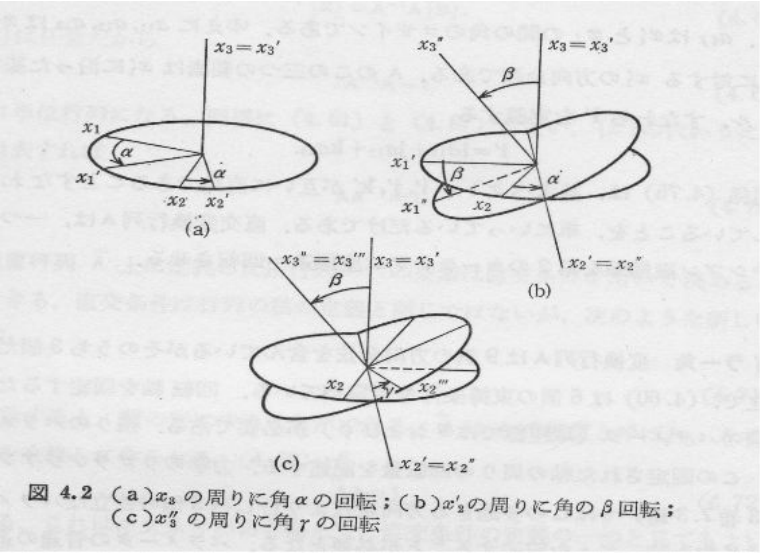

図 2.1: マトリックス回転 [4].

t を角度とすると，

x 軸回りの回転マトリックスは,

$$R_x(t) = \begin{bmatrix} \cos(t) & -\sin(t) & 0 \\ \sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

y 軸回りの回転マトリックスは,

$$R_y(t) = \begin{bmatrix} \cos(t) & 0 & \sin(t) \\ 0 & 1 & 0 \\ -\sin(t) & 0 & \cos(t) \end{bmatrix} \tag{2.2}$$

z 軸回りの回転マトリックスは,

$$R_z(t) = \begin{bmatrix} \cos(t) & -\sin(t) & 0 \\ \sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

と表される.

2.3 ユニットセルの作成

ユニットセルとは，結晶構造の周期パターンの最も小さい単位である.結晶構造は，ユニットセルの敷き詰めで表現されるため，ユニットセルを作成することによって，スーパーセルにも容易に拡げることができる.まず，原子の位置を決定する．原子位置は，データファイルとして読み込んだ原子位置とユニットセルの頂点から伸びる，３本の基本ベクトルを掛け合わせることによって，原子位置のベクトルの大きさと単位ベクトルから原子の位置を決定する.
データファイルを変えることのみによって，3C,4H,6H-SiC の原子位置を決定することが出来た．次に原子の結合 (stick) の座標と角度を求める.まず，ある２つの原子の座標を

$$A = (x_1, x_2, x_3) \tag{2.4}$$

$$B = (y_1, y_2, y_3) \tag{2.5}$$

とする.
次に原子の結合 (stick) の座標と角度を決定する．位置は原子と原子の中

間点が stick の座標となる．
角度を求めるために方向ベクトルを求める．単位ベクトルなる絶対値 1 のベクトルを作れば，それを定数倍して容易に求めるベクトルをだすことが可能となる．
つまり，単位ベクトルを計算し，大きさを規格化したあと，求めたい大きさ倍を n_a とすると，次式のようになる．

$$n_a = \frac{a}{|a|} \tag{2.6}$$

$$a = B - A = \begin{bmatrix} y_1 - x_1 \\ y_2 - x_2 \\ y_3 - x_3 \end{bmatrix} \tag{2.7}$$

なので，

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{2.8}$$

とすると，

$$|a| = \frac{1}{\sqrt{a_1^2 + a_2^2 + a_3^2}} \tag{2.9}$$

となる．よって，

$$n_a = \frac{1}{\sqrt{a_1^2 + a_2^2 + a_3^2}} \times \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{2.10}$$

となる．これで求められたベクトルを，

$$n_a = \begin{bmatrix} X1 \\ Y1 \\ Z1 \end{bmatrix} \tag{2.11}$$

と表記する．
MAYA の特性として，stick を作成したときに x 軸に平行になる．よって，x 軸方向には回転する必要はなく，x 軸の単位ベクトルを y 軸と z 軸の回転マトリックス (式 (3.2),(3.3)) を利用して，そのベクトルが n_a と等しく

なる角度を求めると次のような式になる．

$$\begin{bmatrix} X1 \\ Y1 \\ Y3 \end{bmatrix} = R_z(r1).R_y(r2).(1,0,0) = \begin{bmatrix} \cos(r1)\cos(r2) \\ \sin(r1)\cos(r2) \\ -\sin(r2) \end{bmatrix} \tag{2.12}$$

よって，これを解いて，

$$r2 = -\arcsin(Z1) \tag{2.13}$$

$$r1 = \arccos(X1/\cos(r1)) \tag{2.14}$$

となる．これを，弧度法から度数法に変換すると，

$$R1 = r1 \times (180/\pi) \tag{2.15}$$

$$R2 = r2 \times (180/\pi) \tag{2.16}$$

となる．よって，z 軸を中心に R1 度，y 軸を中心に R2 度回転することがわかる．

第3章　結果

3.1　作成の手順

手法で述べた結晶構造の作成の手順を簡単なフローチャートで現したものが図3.1である．まず，Rubyでballの座標のデータファイルを読み込む．このデータファイルを変えるだけで，SiCの3C,4H,6Hの3つの構造だけではなく，他の結晶をも作成することができる．そして，ballやstickの位置を簡単な線形数学で計算する．アニメーションもRubyで記述し，MELに出力するコードを作成した．RubyはMELに比べてシンプルな文法で，線形代数の計算などが容易になり，より効率的にプログラムを書くことが可能となる．

3.1.1　Rubyのプログラム解説

Rubyのプログラムの解説を行う．まず，パッケージの呼び出しをする．

```
#!/usr/bin/ruby
include Math
require 'yaml'
require 'pp'
require "matrix"
require 'nabe_light_script'
```

'filename1='でテキストファイルを出力する時は"ファイルネーム.txt"というコードを用いてきたが，MELの形式に変換する必要があるので．拡張子を"ファイルネーム.mel"とすることによって，Rubyで記述したものをMELに出力するコードを作成した．また．ballの座標のテキストファイルをinfile = File.read("ファイルネーム")とし，ファイルネーム(データファイル)を換えることで結晶を作ることが可能となった．

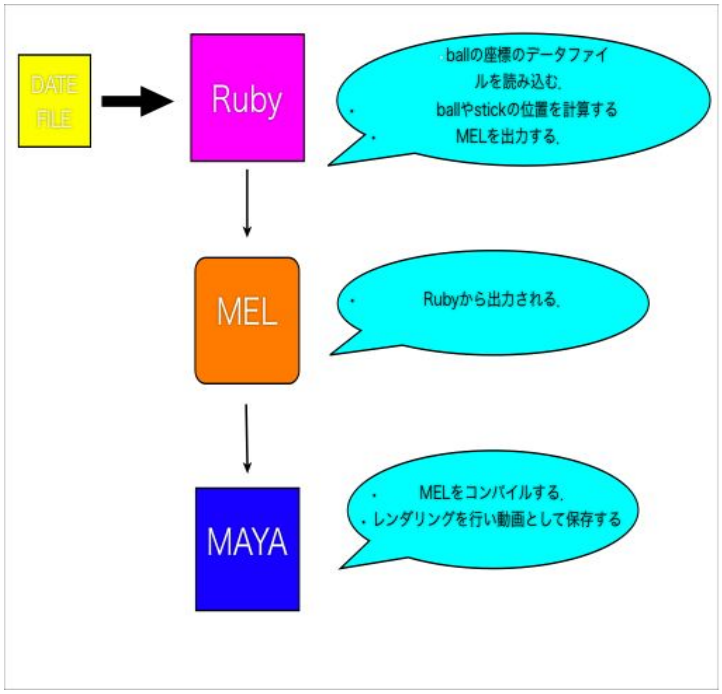

図 3.1: 結晶構造の作成の手順

¿ruby ファイルネーム.rb
を実行すると，ファイルネーム.mel と出力される．

```
filename1="camera_ball_4h.mel"
file1 = File.open(filename1,"w")
infile = File.read("vector_data_4h")
```

原子位置とユニットセルの頂点から伸びる，3本の基本ベクトル(primitive ベクトル) を掛け合わせることによって，原子位置のベクトルの大きさと単位ベクトルから原子の位置を決定するし，色を分類するために Si と C とで別の配列にいれる．

```
p_vec=[]
for i in 0..2
  p1= infile.grep(/a.*$/)[i].split(/=/)[1].chomp.split(/ +/)
  p_vec.push Vector[p1[1].to_f,p1[3].to_f,p1[2].to_f]
end

num1=infile.grep(/Basis Vectors.*$/)[0].split(/:/)[1].to_i/2

si_ball=[]
for i in 0..num1-1 do
  p1=infile.grep(/Si  /)[i].split(/ +/)
  si_ball.push Vector[p1[1].to_f,p1[3].to_f,p1[2].to_f]
end

c_ball=[]
for i in 0..num1-1 do
  p1=infile.grep(/C  /)[i].split(/ +/)
  c_ball.push Vector[p1[1].to_f,p1[3].to_f,p1[2].to_f]
end

ball0=[]
element=[]
#si_c_num=si_ball.size
for i in 0..num1-1 do
  ball0.push si_ball[i]
```

```
  element.push 0
  ball0.push c_ball[i]
  element.push 1
end

ballE=element
ball1=[]
unitcell=[]
for i in 0..ball0.size-1 do
  ball1.push p_vec[0]*ball0[i][0]+p_vec[2]*ball0[i][1]+\\
  p_vec[1]*ball0[i][2]
  unitcell.push p_vec[0]*ball0[i][0]+p_vec[2]*ball0[i][1]+\\
  p_vec[1]*ball0[i][2]
end
```

取得したユニットセルの原子位置をユニットセル単位で繋げることによって，スーパーセルまで拡げる．

```
unitcellE=element
unitcell=ball1
unimax=unitcell.size
n=1
nn=1

def unitcell_plus(ball1,ballE,unitcell,unitcellE,p_vec,n,nn)
  y_vec=Vector[0,10.084,0]
  unimax = unitcell.size

  for p in -n..n do
    for q in -n..n do
      for k in 0..nn do
        for i in 0..unimax-1 do
          if p==0 && q==0 && k==0 then next
          else
            ball1.push unitcell[i]+p_vec[0]*p+p_vec[1]*q+y_vec*k
            ballE.push unitcellE[i]
          end
```

```
        end
      end
    end
  end
  return ball1
end


def unitcell_plus2(ball1,ballE,unitcell,unitcellE,p_vec,n)
  unimax = unitcell.size
  for p in -n..n do
    for q in -n..n do
      for i in 0..unimax-1 do
        if p==0 && q==0 then next
        else
          ball1.push unitcell[i]+p_vec[0]*p+p_vec[1]*q
          ballE.push unitcellE[i]
        end
      end
    end
  end
  return ball1
end

ball1 = unitcell_plus(ball1,ballE,unitcell,unitcellE,p_vec,n,nn)

#y_vec=Vector[0,10.08506484,0]
unimax = ball0.size
p=2
#nn=2
  for q in 0..1 do
    # for p in 0..nn do
    for i in 0..unimax-1 do
      if p==0 && q==0  then next
      else
```

```
        ball1.push unitcell[i]+p_vec[0]*p+p_vec[1]*q
        ballE.push unitcellE[i]
      end
    end
    #end
  end
nmax=ball1.size

uni_num=nmax/ball0.size
```

MEL の色の指定をここで記述する.

```
file1.print <<"EOB"
setAttr \"lambert1.color\" -type double3 1 1 0;

shadingNode -asShader lambert;
sets -renderable true -noSurfaceShader true -empty -name lambert2SG;
connectAttr -f lambert2.outColor lambert2SG.surfaceShader;
select -r lambert2;
setAttr "lambert2.color" -type double3 1 1 1;

EOB

file1.print <<"EOB"
shadingNode -asShader lambert;
sets -renderable true -noSurfaceShader true -empty -name lambert3SG;
connectAttr -f lambert3.outColor lambert3SG.surfaceShader;
select -r lambert3;
setAttr \"lambert3.color\" -type double3 0.4 0.4 0.4;

EOB

file1.print <<"EOB"
shadingNode -asShader lambert;
sets -renderable true -noSurfaceShader true -empty -name lambert4SG;
connectAttr -f lambert4.outColor lambert4SG.surfaceShader;
select -r lambert4;
```

```
setAttr "lambert4.color" -type double3 1 0 1 ;
setAttr "lambert4.transparency" -type double3 0.5 0.5 0.5 ;

EOB

file1.print <<"EOB"
shadingNode -asShader lambert;
sets -renderable true -noSurfaceShader true -empty -name lambert5SG;
connectAttr -f lambert5.outColor lambert5SG.surfaceShader;
select -r lambert5;
setAttr "lambert5.color" -type double3 1 1 1 ;

EOB



file1.print <<"EOB"
shadingNode -asShader lambert;
sets -renderable true -noSurfaceShader true -empty -name lambert6SG;
connectAttr -f lambert6.outColor lambert6SG.surfaceShader;
select -r lambert6;
setAttr "lambert6.color" -type double3 1 0 0 ;

EOB

file1 = nabe_light(file1,0.6)
```

MEL のプロシージャーの記述である.

```
file1.print <<"EOB"
global proc make_Si(float $x,float $y,float $z)
{
sphere;
scale 0.5 0.5 0.5;
move $x $y $z;
}
global proc make_C(float $x,float $y,float $z)
```

```
{
sphere;
scale 0.5 0.5 0.5;
move $x $y $z;
sets -e -forceElement lambert3SG;
}
global proc make_O(float $x,float $y,float $z)
{
sphere;
scale 50 50 50;
move $x $y $z;
sets -e -forceElement lambert5SG;
}
global proc make_cylinder(float $x,float $y,float $z,float $b2,
float $a2)
{
cylinder;
scale 1.0 0.2 0.2;
move $x $y $z;
rotate 0 $b2 $a2;
sets -e -forceElement lambert2SG;
}
global proc make_wireframe(float $x,float $y,float $z,float $b2,
float $a2,float $sa)
{
cylinder;
scale $sa 0.05 0.05;
move $x $y $z;
rotate 0 $b2 $a2;
sets -e -forceElement lambert6SG;
}
EOB
```

別々の配列に入っている Si と C の座標位置から記述を行う.

```
n_ballE=ballE.size
for i in 0..n_ballE-1 do
```

```
  if(ballE[i]==0)
    file1.printf("make_Si(%7.5f,%7.5f,%7.5f);\n",ball1[i][0],ball1[i][1],
    ball1[i][2])
  else
    file1.printf("make_C(%7.5f,%7.5f,%7.5f);\n",ball1[i][0],ball1[i][1],
    ball1[i][2])
  end
end
```

原子間の距離が最も近い ball の組み合わせを探し，その座標を配列に入れる．

```
comb_tmp=[]
for i in 0..nmax-1 do
  for j in i+1..nmax-1 do
    tmp_vector=ball1[i]-ball1[j]
    if tmp_vector.r < 2.0 then
      comb_tmp.push([i,j])
    end
  end
end
```

見つかった原子の組み合わせの配列の座標から 3 次元マトリックスを利用して stick の座標と角度を求める．MAYA の特徴として，y 軸のベクトルの計算で，大きい方から小さい方をひかなければ正しい数値が得られないため，if 文で場合分けをした．MAYA は弧度法ではなく，度数法のため変換している．stick の太さは試行錯誤の結果，SiC のような 4 配位の結晶の場合，理解しやすかった 0.2 にした．そして，得られた座標位置，角度から，stick の記述をする．

```
comb_tmp.each{|ij|
  half_v=(ball1[ij[0]]+ball1[ij[1]])*0.5
  tmp=ball1[ij[0]]-ball1[ij[1]]
  if tmp[1] < 0  then
    dir_vec=ball1[ij[1]]-ball1[ij[0]]
  else
    dir_vec=ball1[ij[0]]-ball1[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(%7.5f,%7.5f,%7.5f,%7.5f,
    %7.5f,\n,half_v[0],half_v[1],half_v[2],b2,a2)
}
```

primitive ベクトルはユニットセルを囲むベクトルなので，primitive ベクトルの座標から wire を作成する．wire の角度は stick の角度を求める方法と同様である．wire の色を stick より際立つ色にし，太さも細くすることによってユニットセルを浮き上がって見ることが出来るようにした．

```
def wire_frame(p_vec,file1)
  toga = [ Vector[0.0, 0.0, 0.0],
         Vector[p_vec[0][0],p_vec[0][1],p_vec[0][2]],
         Vector[p_vec[1][0],p_vec[1][1],p_vec[1][2]],
         Vector[p_vec[2][0],p_vec[2][1],p_vec[2][2]],
         Vector[p_vec[0][0]+p_vec[1][0],p_vec[0][1]+p_vec[1][1],\\
         p_vec[0][2]+p_vec[1][2]],
         Vector[p_vec[0][0]+p_vec[2][0],p_vec[0][1]+p_vec[2][1],\\
         p_vec[0][2]+p_vec[2][2]],
         Vector[p_vec[1][0]+p_vec[2][0],p_vec[1][1]+p_vec[2][1],\\
   p_vec[1][2]+p_vec[2][2]],
         Vector[p_vec[0][0]+p_vec[1][0]+p_vec[2][0],\\
  p_vec[0][1]+p_vec[1][1]+p_vec[2][1],p_vec[0][2]+p_vec[1][2]
       +p_vec[2][2]]]


  co = [[0,1],[0,2],[0,3],[1,4],[1,5],[2,4],
```

```
        [2,6],[3,5],[3,6],[4,7],[5,7],[6,7]]
  for i in 0..co.length-1 do
    half_v = (toga[co[i][0]]+toga[co[i][1]])*0.5
    tmp = toga[co[i][0]]-toga[co[i][1]]
    if tmp[1] < 0  then
      #dir_vec=toga[co[i][1]]-toga[co[i][1]]
      dir_vec = tmp*(-1)
    else
      dir_vec=tmp
    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_wireframe(%7.5f,%7.5f,%7.5f,%7.5f,%7.5f,%7.5f);\\
   \n",half_v[0],half_v[1],half_v[2],b2,a2,(1/aa)/2)
  end
end

wire_frame(p_vec,file1)
```

面を挿入し，挿入されたところを指定し，切り離し，カメラを分かりやすい位置まで移動させるアニメーションの作成である．カメラは結晶の回りを回転し，斜め上からも見えるようにプログラムした．

```
file1.print <<"EOB"
select -r nurbsSphere18 nurbsSphere19 nurbsSphere20 nurbsSphere21
nurbsSphere22 nurbsSphere23 nurbsSphere24 nurbsSphere34
nurbsSphere35 nurbsSphere36 nurbsSphere37 nurbsSphere38 nurbsSphere39
nurbsSphere40 nurbsSphere50 nurbsSphere51 nurbsSphere52
```

```
nurbsSphere53 nurbsSphere54 nurbsSphere55 nurbsSphere56
nurbsSphere66 nurbsSphere67 nurbsSphere68 nurbsSphere69
nurbsSphere70 nurbsSphere71 nurbsSphere72 nurbsSphere74
nurbsSphere75 nurbsSphere76 nurbsSphere77 nurbsSphere78
nurbsSphere79 nurbsSphere80 nurbsSphere90 nurbsSphere91
nurbsSphere92 nurbsSphere93 nurbsSphere94 nurbsSphere95
nurbsSphere96 nurbsSphere106 nurbsSphere107 nurbsSphere108
nurbsSphere109 nurbsSphere110 nurbsSphere111 nurbsSphere112
nurbsSphere122 nurbsSphere123 nurbsSphere124 nurbsSphere125
nurbsSphere126 nurbsSphere127 nurbsSphere128 nurbsSphere138
nurbsSphere139 nurbsSphere140 nurbsSphere141 nurbsSphere142
nurbsSphere143 nurbsSphere144 nurbsCylinder36 nurbsCylinder37
nurbsCylinder38 nurbsCylinder39 nurbsCylinder40 nurbsCylinder41
nurbsCylinder42 nurbsCylinder43 nurbsCylinder44 nurbsCylinder45
nurbsCylinder46 nurbsCylinder47 nurbsCylinder48 nurbsCylinder64
nurbsCylinder65 nurbsCylinder66 nurbsCylinder67 nurbsCylinder68
nurbsCylinder69 nurbsCylinder70 nurbsCylinder71 nurbsCylinder72
nurbsCylinder73 nurbsCylinder74 nurbsCylinder75 nurbsCylinder76
nurbsCylinder87 nurbsCylinder88 nurbsCylinder89 nurbsCylinder90
nurbsCylinder91 nurbsCylinder92 nurbsCylinder93 nurbsCylinder94
nurbsCylinder109 nurbsCylinder110 nurbsCylinder111

nurbsCylinder112nurbsCylinder113 nurbsCylinder114
nurbsCylinder115 nurbsCylinder116 nurbsCylinder117
nurbsCylinder118 nurbsCylinder119 nurbsCylinder120
nurbsCylinder121 nurbsCylinder122 nurbsCylinder123
nurbsCylinder124 nurbsCylinder125 nurbsCylinder126
nurbsCylinder127 nurbsCylinder128 nurbsCylinder129
nurbsCylinder130 nurbsCylinder131 nurbsCylinder132
nurbsCylinder133 nurbsCylinder134 nurbsCylinder145
nurbsCylinder146 nurbsCylinder147 nurbsCylinder148
nurbsCylinder149 nurbsCylinder150 nurbsCylinder151
nurbsCylinder152 nurbsCylinder166 nurbsCylinder167
nurbsCylinder168 nurbsCylinder169 nurbsCylinder170
nurbsCylinder171 nurbsCylinder172 nurbsCylinder173
```

```
nurbsCylinder174 nurbsCylinder189 nurbsCylinder190
nurbsCylinder191 nurbsCylinder192 nurbsCylinder193
nurbsCylinder194 nurbsCylinder195 nurbsCylinder196
nurbsCylinder197 nurbsCylinder207 nurbsCylinder208
nurbsCylinder209 nurbsCylinder210 nurbsCylinder211
nurbsCylinder212 nurbsCylinder213 ;
group;
camera -centerOfInterest 5 -focalLength 35 -lensSqueezeRatio 1 \\
-cameraScale 1 -horizontalFilmAperture 1.41732 -horizontal\\
FilmOffset 0 -\verticalFilmAperture 0.94488 -verticalFilmOffset 0\\
 -filmFit Fill -overscan 1 -motionBlur 0 -shutterAngle 144\\
 -nearClipPlane 0.01 -farClipPlane 1000 -orthographic 0 \\
-orthographicWidth 30; \\
objectMoveCommand;
cameraMakeNode 2 "";
setAttr "camera1_aim.translateX" 0;
setAttr "camera1_aim.translateY" 5;
setAttr "camera1_aim.translateZ" 0;
setAttr"camera1.translateX" 0;
setAttr"camera1.translateY" 20;
setAttr"camera1.translateZ" 25;
select -r group1;
string $pnames[] = `ls -sl`;
select -r  camera1;
string $tnames[] = `ls -sl`;
EOB



file1.print <<"EOB"
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];
}
EOB


time = 0

#for i in 0..20 do
25.times do |i|
  file1.printf("make_current($pnames,%d,0,%d,0);\n",time,i)
  if(time==1)
    file1.printf("make_current($tnames,%d,0,11,25);\n",time)
  end
  i=i+25
  time += 1
end


file1.print <<"EOB"
string $pnames[];
$pname = `nurbsPlane`;
sets -e -forceElement lambert4SG;
scale 15 15 15;
select -r nurbsPlane1;
string $snames[] = `ls -sl`;

EOB


i=30
#for j in 21..36 do
j = time+1
```

```
16.times do |j|
  while (i>4)
    file1.printf("make_current($snames,%d,3.081,%d,-1);\n",j,i)
    #file1.printf("make_current($tnames,%d,0,11,35);\n",j)-3.557
    i=i-5
    j=j+5
  end
  time += 1
end



file1.print <<"EOB"
select -r nurbsSphere133 nurbsSphere134 nurbsSphere135 nurbsSphere136 \\
nurbsSphere149 nurbsSphere150 nurbsSphere151 nurbsSphere152 \\
nurbsSphere153 nurbsSphere154 nurbsSphere155 nurbsSphere156 \\
nurbsSphere157 nurbsSphere158 nurbsSphere159 nurbsSphere160\\
nurbsCylinder182 nurbsCylinder184 nurbsCylinder187 nurbsCylinder199 \\
nurbsCylinder201 nurbsCylinder202 nurbsCylinder203 nurbsCylinder204 \\
nurbsCylinder205 nurbsCylinder206 nurbsCylinder215 nurbsCylinder217 \\
nurbsCylinder218 nurbsCylinder219 nurbsCylinder220 nurbsCylinder221 \\
 nurbsCylinder222 nurbsCylinder223 nurbsCylinder224 nurbsCylinder225 \\
nurbsCylinder226 nurbsCylinder227 nurbsCylinder228 nurbsCylinder229 ; \\
group;
select -r group2;
string $qnames[] = `ls -sl`;

EOB

a=0
#for k in 37..80 do
30.times do |i|
  if(i==0 || i==29)
    file1.printf("make_current($qnames,%d,%d,0,0);\n",time,a)
    if(i==29)
```

```
    file1.printf("make_current($tnames,%d,0,11,25);\n",time)
    end
  end
  a=a+3
  time += 1
end

l=0
m=71
#for m in 81..130 do
56.times do |i|
  z=25*cos(l*PI/180)
  x=25*sin(l*PI/180)
  file1.printf("make_current($tnames,%d,%7.5f,11,%7.5f);\n",time,x,z)
  if(i<20 || i>40)
    l=l+5
  end
  time += 1
end

p=11
#for n in 131..150 do
36.times do |i|
  z=25*cos(l*PI/180)
  x=25*sin(l*PI/180)
  file1.printf("make_current($tnames,%d,%7.5f,%7.5f,%7.5f);\n",time,x,p,z)
  if(i<18)
    p=p+1
  else
    p -= 1
  end
  l=l+5
  time += 1
end
```

```
20.times do |i|
  z=25*cos(l*PI/180)
  x=25*sin(l*PI/180)
  file1.printf("make_current($tnames,%d,%7.5f,11,%7.5f);\n",time,x,z)
 # if(i<20 || i>40)
    l=l+5
#  end
  time += 1
end
p time



file1.close
exit!
```

3.1.2 MEL のプログラム解説

MEL のプログラムの一例を示す.
最初の色の指定である．すべての対象が色を指定しなければこの色になる．MAYA は赤と緑と青の比率によって色を作成する．次の場合 1 1 0 なので黄色になる．lambert はサーフェースを作成するマテリアルシェーダーである.

```
setAttr "lambert1.color" -type double3 1 1 0;
```

これで次の色を作成する．まず，lambert を作成し，名前をつけ，その lambert に色をつける．2 色目以降は全て同様である.

```
shadingNode -asShader lambert;
sets -renderable true -noSurfaceShader true -empty -name lambert2SG;
connectAttr -f lambert2.outColor lambert2SG.surfaceShader;
select -r lambert2;
setAttr "lambert2.color" -type double3 1 1 1;
```

プロシージャーにすることによって，プロシージャーに変更点がなければ何回でも同じスクリプトをプロシージャー名で実行出来る．Si,C,cylinder,wireframeで分けている．また．プロシージャーにすれば，スクリプトの量も軽減することが可能となる．

```
global proc make_Si(float $x,float $y,float $z)
{
sphere;
scale 0.5 0.5 0.5;
move $x $y $z;
}
global proc make_C(float $x,float $y,float $z)
{
sphere;
scale 0.5 0.5 0.5;
move $x $y $z;
sets -e -forceElement lambert3SG;
}
global proc make_cylinder(float $x,float $y,float $z,float $b2,float $a2)
{
cylinder;
scale 1.0 0.2 0.2;
move $x $y $z;
rotate 0 $b2 $a2;
sets -e -forceElement lambert2SG;
}
global proc make_wireframe(float $x,float $y,float $z,float $b2,\\
float $a2,float $sa)
{
cylinder;
scale $sa 0.05 0.05;
move $x $y $z;
rotate 0 $b2 $a2;
sets -e -forceElement lambert6SG;
}
```

次のようにプロシージャー名を換えて，プロシージャーを呼び出し，x 軸,y 軸,z 軸の座標を入力する．

```
make_Si(0.00000,0.00000,0.00000);
make_C(0.00000,1.89090,0.00000);
make_cylinder(0.00000,0.94545,0.00000,-0.00000,90.00000);
make_wireframe(1.54025,5.04240,-2.66780,0.00000,90.00000,5.04240);
```

3.1.3 データファイル

Ruby で記述するときに読み込む原子の座標のデータファイルである．$Primitivevectors$ と $BasisVectors$ のファイルを表示したい原子のデータに換えるだけでその原子の結晶を表示することが出来る．

```
Primitive vectors
a(1) =   1.54025500 -2.66779992  0.00000000
a(2) =   1.54025500  2.66779992  0.00000000
a(3) =   0.00000000  0.00000000 10.08480000


Volume =  82.87874525


Reciprocal vectors
b(1) =   0.32462157 -0.18742035  0.00000000
b(2) =   0.32462157  0.18742035  0.00000000
b(3) =   0.00000000  0.00000000  0.09915913


Basis Vectors:8
Atom    Lattice Coordinates                Cartesian Coordinates

Si  0.00000000  0.00000000  0.00000000     0.00000000  0.00000000 \\
 0.00000000
Si  0.00000000  0.00000000  0.50000000     0.00000000  0.00000000 \\
 5.04240000
```

```
C   0.00000000  0.00000000  0.18750000     0.00000000  0.00000000 \\
1.89090000
C   0.00000000  0.00000000  0.68750000     0.00000000  0.00000000 \\
6.93330000
Si  0.33333333  0.66666667  0.24982500     1.54025500  0.88926664 \\
 2.51943516
Si  0.66666667  0.33333333  0.74982500     1.54025500 -0.88926664 \\
 7.56183516
C   0.33333333  0.66666667  0.43732500     1.54025500  0.88926664 \\
 4.41033516
C   0.66666667  0.33333333  0.93732500     1.54025500 -0.88926664 \\
 9.45273516
```

3C-SiC のデータファイル.

```
Primitive vectors
a(1) =   4.36 0.00000000  0.00000000
a(2) =   0.00000000  0.00000000  4.36
a(3) =   0.00000000  4.36 0.00000000


Volume =  82.87874525


Reciprocal vectors
b(1) =   0.32462157 -0.18742035  0.00000000
b(2) =   0.32462157  0.18742035  0.00000000
b(3) =   0.00000000  0.00000000  0.09915913


Basis Vectors:8
Atom    Lattice Coordinates                Cartesian Coordinates

Si  0.25  0.25  0.25     0.00000000  0.00000000  0.00000000
Si  0.75  0.25  0.75     0.00000000  0.00000000  5.04240000
C   0.00000000  0.00000000  0.00000000    0.00000000  0.00000000 \\
 1.89090000
```

```
C   0.50000000  0.00000000  0.50000000     0.00000000  0.00000000 \\
 6.93330000
Si  0.75  0.75  0.25     1.54025500  0.88926664  2.51943516
Si  0.25  0.75  0.75     1.54025500 -0.88926664  7.56183516
C   0.50000000  0.50000000  0.00000000     1.54025500  0.88926664 \\
 4.41033516
C   0.00000000  0.50000000  0.50000000     1.54025500 -0.88926664 \\
 9.45273516
```

6H-SiC のデータファイルである.

```
Primitive vectors
a(1) =   1.54035000 -2.66796446   .00000000
a(2) =   1.54035000  2.66796446   .00000000
a(3) =    .00000000   .00000000 15.11740000


Volume = 124.25290558


Reciprocal vectors
b(1) =    .32460155  -.18740879   .00000000
b(2) =    .32460155   .18740879  -.00000000
b(3) =   -.00000000   .00000000   .06614894


Basis Vectors:12
Atom    Lattice Coordinates                Cartesian Coordinates

C    .00000000   .00000000   .12540000      .00000000   .00000000 \\
1.89572196
C    .00000000   .00000000   .62540000      .00000000   .00000000 \\
 9.45442196
Si   .00000000   .00000000   .00000000      .00000000   .00000000  \\
 .00000000
Si    .00000000   .00000000   .50000000      .00000000   .00000000 \\
 7.55870000
```

```
C     .33333333    .66666667    .79190000      1.54035000    .88932149 \\
11.97146906
C     .66666667    .33333333    .29190000      1.54035000  -.88932149 \\
 4.41276906
C     .33333333    .66666667    .45840000      1.54035000    .88932149 \\
 6.92981616
C     .66666667    .33333333  -.04160000      1.54035000  -.88932149 \\
 -.62888384
Si    .33333333    .66666667    .66670000      1.54035000    .88932149 \\
10.07877058
Si    .66666667    .33333333    .16670000      1.54035000  -.88932149 \\
 2.52007058
Si    .33333333    .66666667    .33320000      1.54035000    .88932149 \\
 5.03711768
Si    .66666667    .33333333  -.16680000      1.54035000  -.88932149 \\
-2.52158232
```

3.2 結晶構造の違い

図 3.2 は文献などによく載っている，3C,4H,6H-SiC の結晶構造を (11-20) 面から見た模式図である．
3C-SiC は下から順番に A,B,C と周期的になっている．
6H-SiC は下から順番に A,B,C,A,C,B と周期的になってる．
4H-SiC は下から順番に A,B,A,C と周期的になっている．
SiC は 3C,4H,6H のどれも局所構造は同じで 4 配位で同じだが，積層構造が違うためこのように差が出来る．しかし，この模式図を見ただけでは，初学者にとって，この図が何を意図しているのか理解するのは容易ではない．そこで，今回の研究で，MAYA で視覚化することによって，容易に理解することが可能となった．
MAYA では，切断したいところに面を挿入し，その面の透明度を上げることによって，切断した面から見た時に立体的に捉えることを容易にした．それによって，切断したところの ball と stick が浮き上がって見え，どのような構造をとっているか理解しやすいようになった．

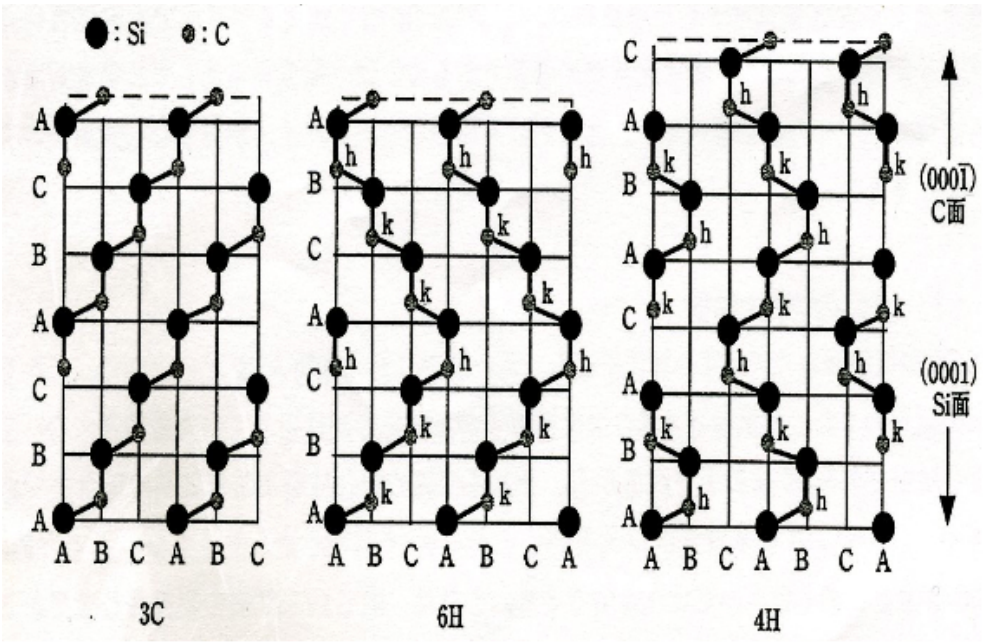

図 3.2: 3c,4h,6h の結晶構造 ((11-20) 面から見た) 模式図.

3.3 MAYA による SiC の結晶構造

3.3.1 3C,4H,6H-SiC

図 3.3 は 3C-SiC の結晶構造を示している.3C-SiC は,4H,6H とは違い, 炭素原子が 3 周期の立方晶である. 黄色の ball が Si で黒い ball が C で白い棒が原子間の結合を表している. また, 赤線で囲まれているのが立方晶を示すユニットセルである.
図 3.4 は 4H-SiC の結晶構造を示している.4H-SiC は, 炭素原子が 4 周期の立方晶である. 赤線で囲まれているのがユニットセルである.
図 3.5 は 6H-SiC の結晶構造を示している.6H-SiC は, 炭素原子が 6 周期の立方晶である. 赤線で囲まれているのがユニットセルである.

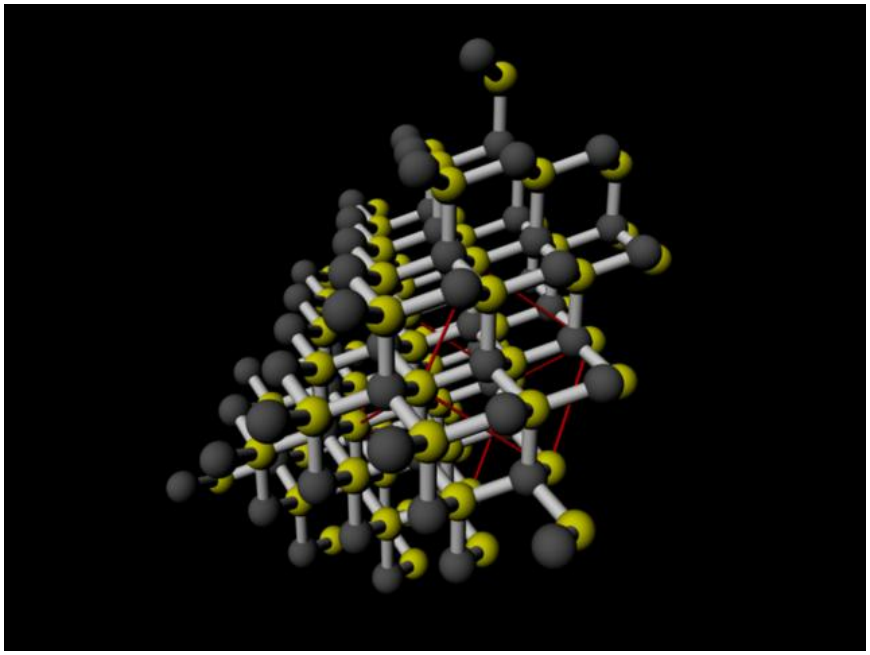

図 3.3: 3C の結晶構造

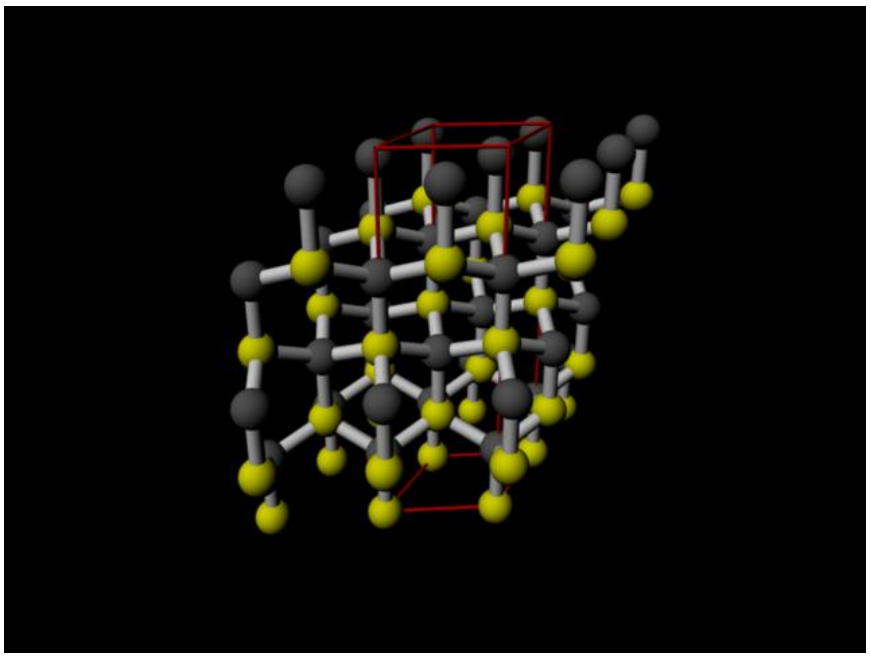

図 3.4: 4H の結晶構造

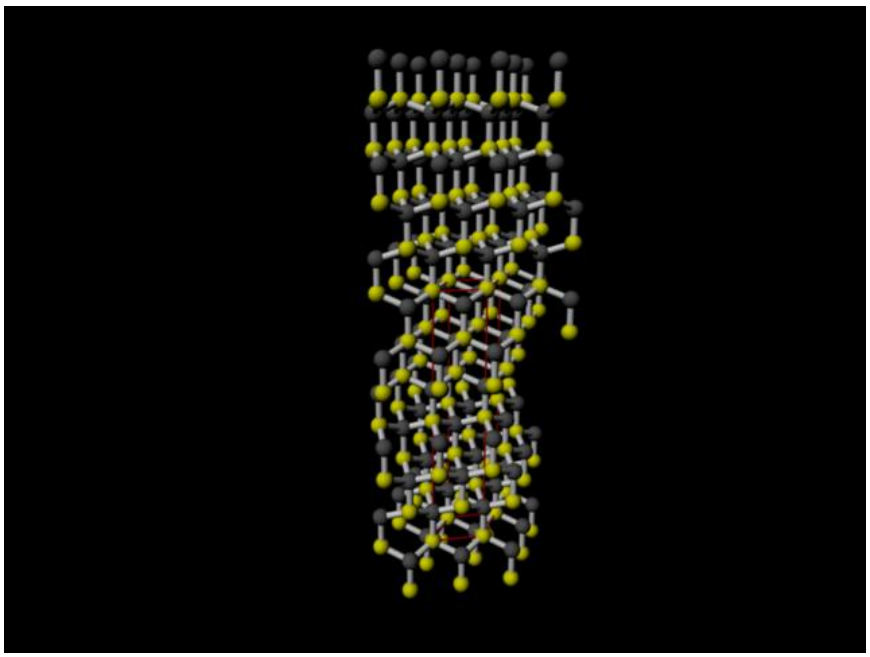

図 3.5: 6H の結晶構造

3.4 MAYA による SiC の結晶表面

図 3.6 は，4H-SiC を上から見た図である．4H-SiC は，六方晶なので図 3.6 のように六角形が見つけられる．6h-SiC も同様に六方晶なので，同じように六角形が見える．
(11-20) 面は，図 3.6 の (a-e)，もしくは，(b-d) を結び，垂直に切断した面である．
(0001) 面は，図 3.6 の状態で結晶の六角形が見れる面である．SiC の場合，Si と C が交互に積み重なっているため，Si 面と C 面が出来る．
(1-100) 面は，図 3.6 の (b-c)，もしくは，(e-f) を結び，垂直に切断した面である．

3.4.1 3C,4H,6H-SiC の (11-20) 面での切断表面

3C-SiC は，4H-SiC,6H-SiC と違い，立方晶であるため，面の取り方が異なる．しかし，立方晶の (11-2) 面は六方晶での (11-20) 面と対応してい

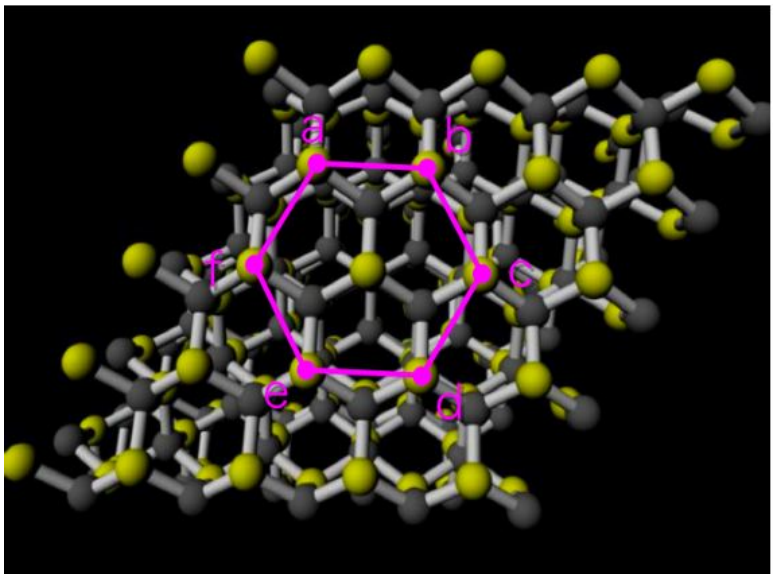

図 3.6: 4H-SiC を (0001) 面から見た図 (六方晶).

るため，(11-20) 面で面をとった．[6]
赤い線で囲まれているユニットセルが立方晶をとっているため，傾いていることが分かる．
図 3.7 に示す．図 3.2 と比較して，同じように，A,B,C となっているが，明らかに MAYA で視覚化した方が理解しやすい．
図 3.8 は，4H-SiC を (11-20) 面で切断し，(11-20) 面から見た図である．
図 3.2 と比較して，A,B,A,C と周期的になっていることが理解出来る．
図 3.9 は，6H-SiC を (11-20) 面で切断し，(11-20) 面から見た図である．周期的に積み重なっていることが分かりやういようにユニットセルを 2 つ積み重ねた．
図 3.2 と比較して，A,B,C,A,C,B と周期的になっていることが理解出来る．

3.4.2 3C,4H,6H-SiC の (0001) 面での切断表面

(11-2) 面の時と同様に，立方晶の (111) 面は六方晶での (0001) 面に対応しているため，(0001) 面で面をとった．図 3.10 に示す．
SiC は局所構造は等しいので，3C-SiC も Si 面と C 面が交互に積み重なっ

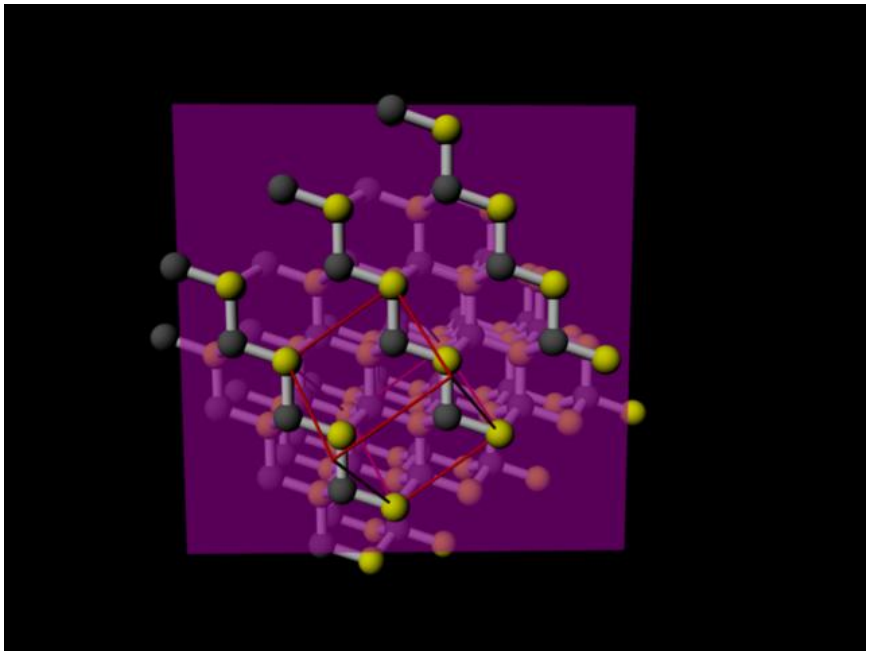

図 3.7: 3C-SiC の (11-20) 面での結晶断面.

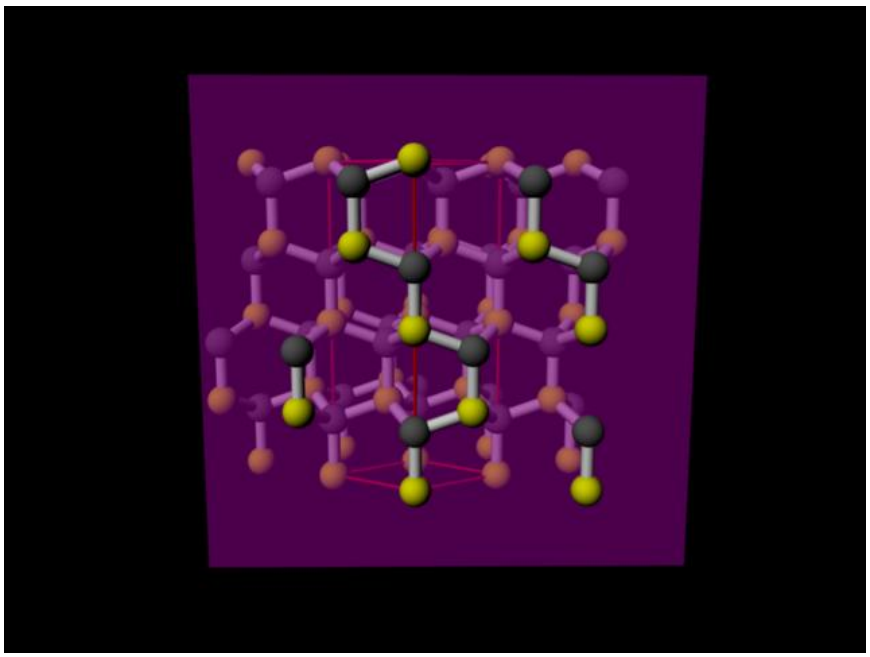

図 3.8: 4H-SiC の (11-20) 面での結晶断面.

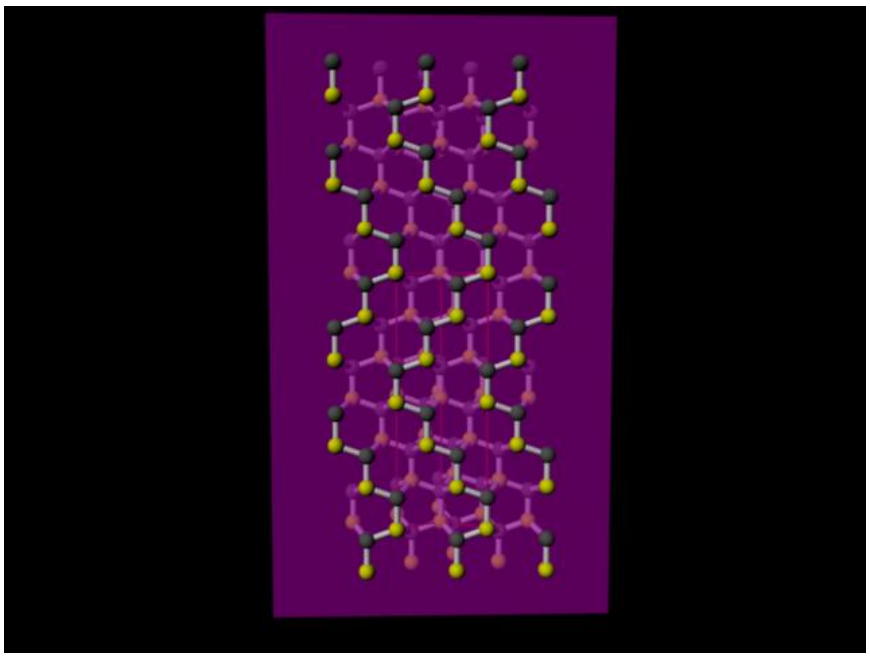

図 3.9: 6H-SiC の (11-20) 面での結晶表面.

ている．図 3.10 の場合は．C 面が見えているが，反対から見ると Si 面が見える．
図 3.11 は 4H-SiC の (0001) 面で切断し，(0001) 面から見たものである．図 3.11 は，C 面が見えていますが，3C-SiC と同様に，反対から見ると Si 面が見える．
図 3.12 は 6H-SiC の (0001) 面で切断し，(0001) 面から見たものである．図 3.12 も，同様に，C 面が見えていますが，反対から見ると，Si 面が見える．

3.4.3 3C,4H,6H-SiC の (1-100) 面での切断表面

先ほどまでと同じように，立方晶の (110) 面は六方晶での (1-100) 面に対応しているため，(1-100) 面で面をとった．図 3.13 に示す．
図 3.13 から，A,B と周期的になっていることが分かる．
図 3.14 は，4H-SiC を (1-100) 面で切断し，(1-100) 面から見た図である．これも，A,B と周期的になっていることが分かる．

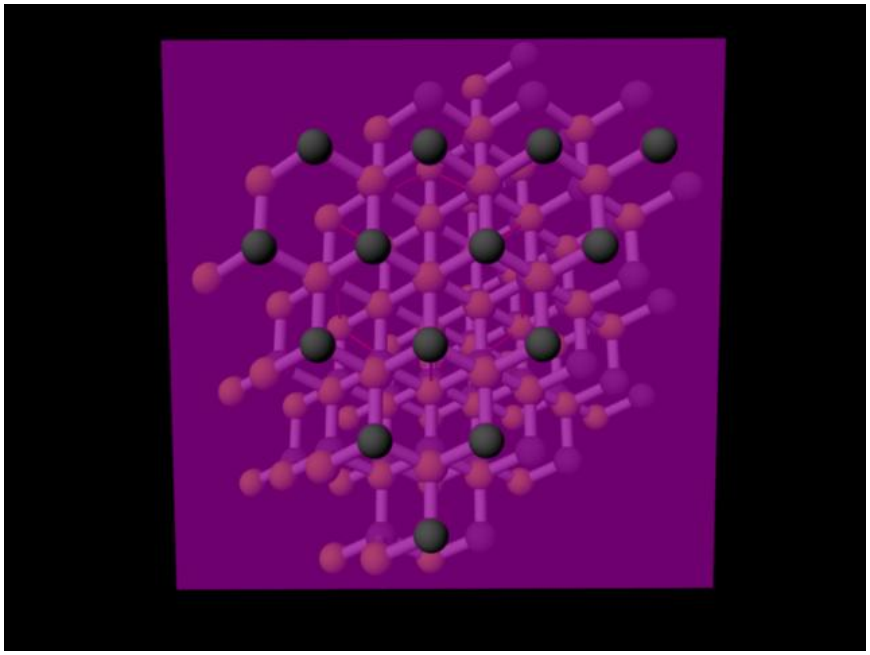

図 3.10: 3C-SiC の (0001) 面での結晶断面.

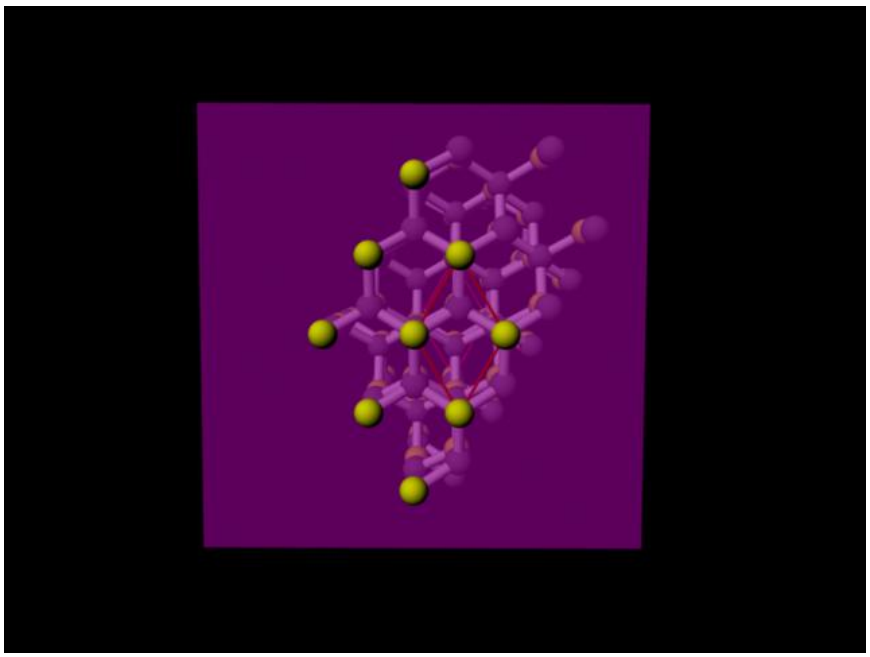

図 3.11: 4H-SiC の (0001) 面での結晶断面.

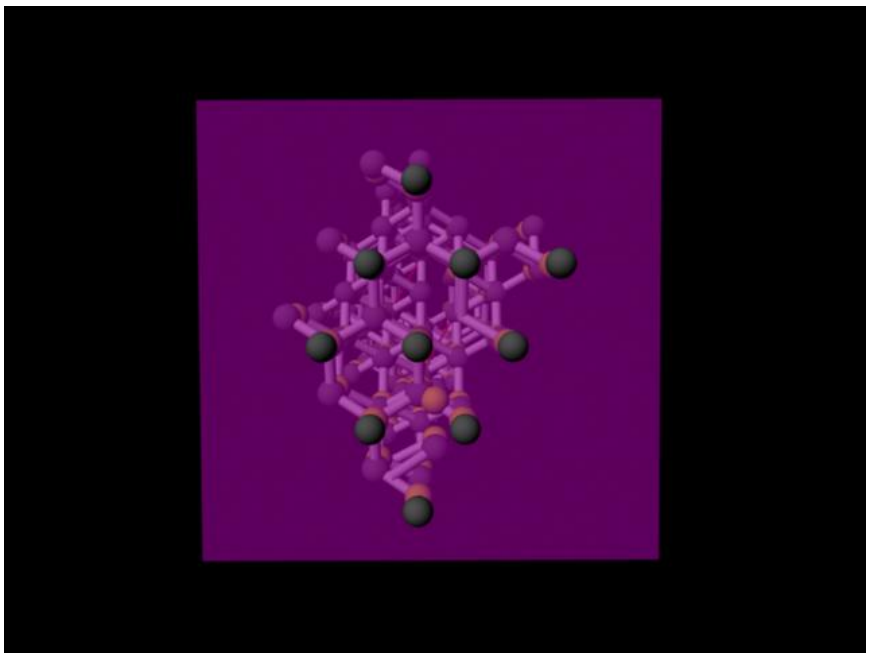

図 3.12: 6H-SiC の (0001) 面での結晶断面.

図 3.15 は，6H-SiC を (1-100) 面で切断し，(1-100) 面から見た図である．これも，A,B と周期的になっていることが分かる．
よって，(1-100) 面は，3C,4H,6H-SiC のどれも A,B と周期的になっていることが分かった.

3.5 Ti の結晶構造

Ti の結晶構造は hcp 構造であるため，12 配位と SiC よりも多いため，stick の太さを細くした．そして，原子位置のデータファイルを換えるだけで Ti の結晶構造を視覚化が可能となった.
図 3.16 に示す.
しかし，この作業は手作業でしているため，配位数と stick の太さを自動

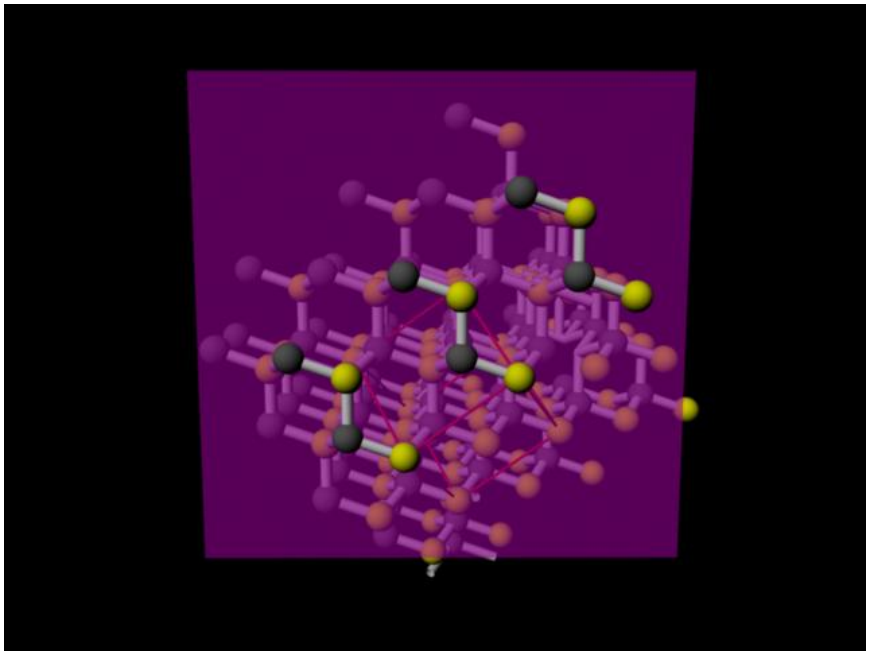

図 3.13: 3C-SiC の (1-100) 面での結晶断面.

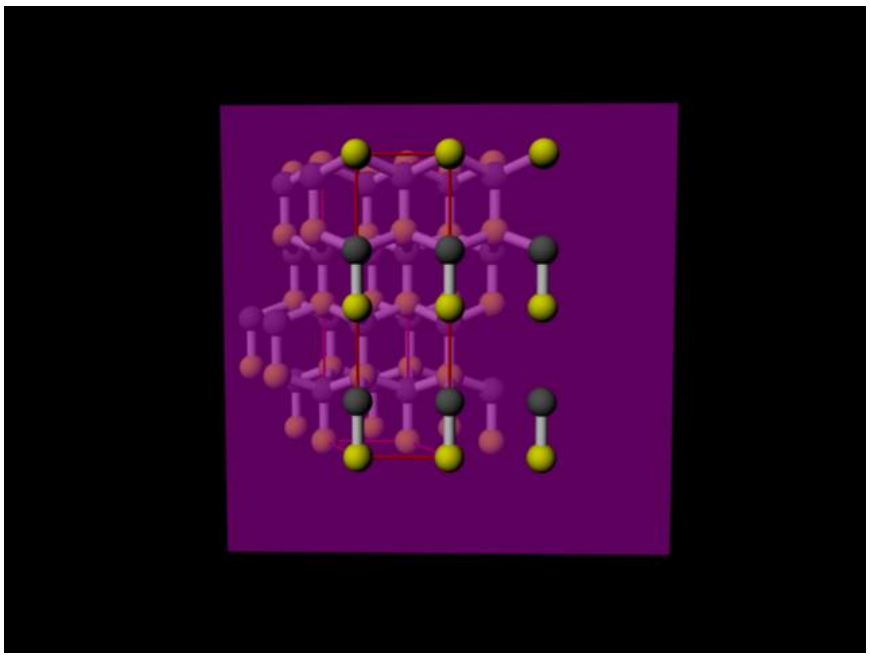

図 3.14: 4h-SiC の (1-100) 面での結晶断面.

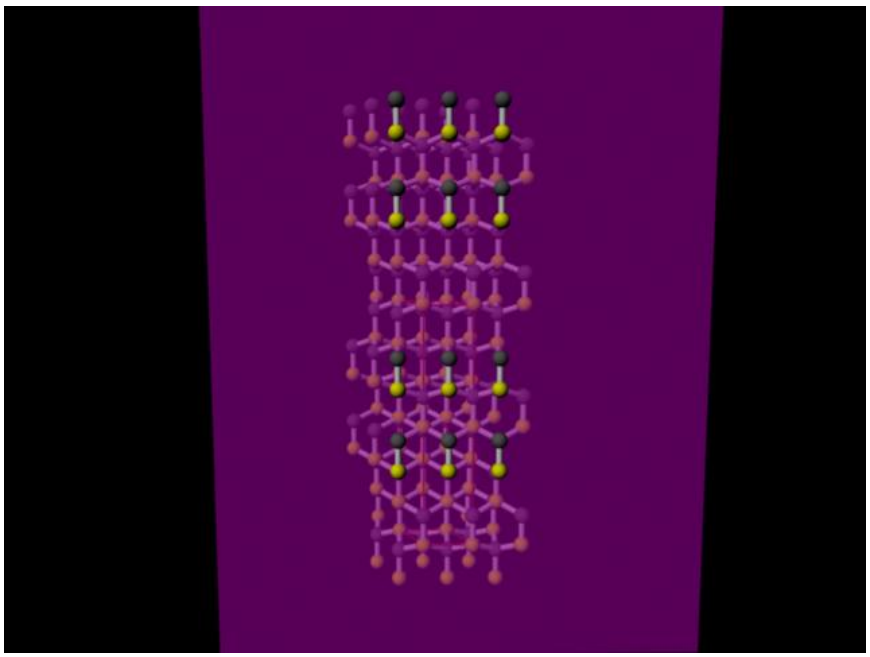

図 3.15: 6h-SiC の (1-100) 面での結晶断面.

的に調整することが課題である.

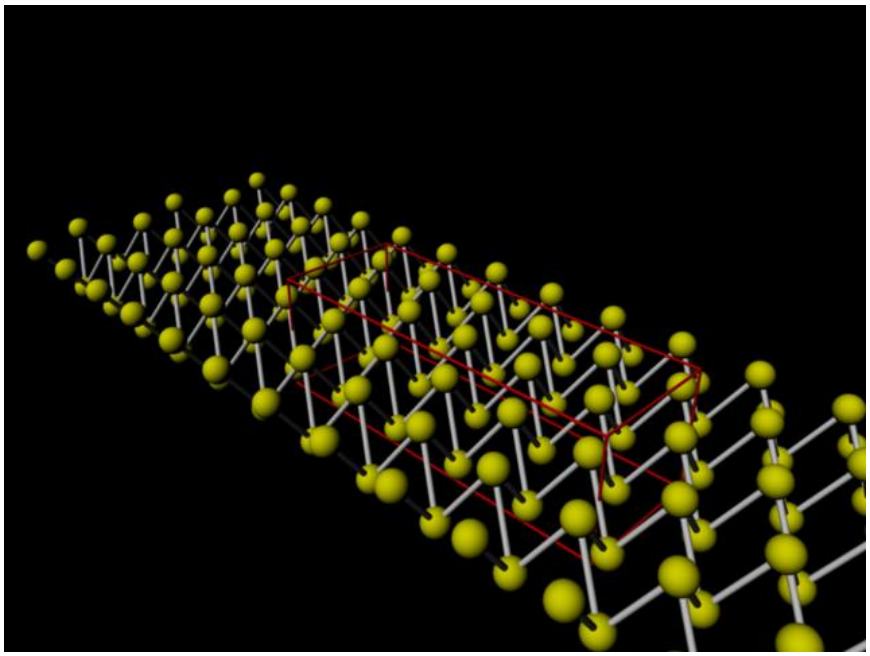

図 3.16: Ti の結晶構造.

第4章　総括

SiC の結晶構造，結晶表面を MAYA で視覚化を試みた．結果は以下の通りである．

- スクリプトの記述の際，Ruby を用いて，MEL を出力するコードを作成した．従来プログラムは MAYA のスクリプト言語である MEL で直接記述するものであった．しかし，MEL はグラフィックスに特化しており，一般的な言語に比べて線形代数の計算などが複雑であった．Ruby は文法がシンプル，コンパイルが不要，エディタとして Emacs が使用出来るなどの利点がある．よって，Ruby で記述することで，より効率的にプログラムを書くことが出来た．
- SiC だけでなく，様々な結晶を視覚化することが可能となった．結晶の原子位置の座標をテキストファイルから読み込む形式をとった．テキストファイルの情報から，原子位置，原子の結合の座標，傾きを計算する．それによって，テキストファイルを換えるだけで，その結晶を作成することが出来た．
- 結晶の格子構造や切断面構造等を視覚化出来るようになった．結晶格子だけであれば，MedeA など既存のソフトで視覚化可能である．しかし，表面や積層欠陥などの切断面はそのような既存のソフトでは表示が困難であった．MAYA では，切断したいところに面を挿入し，その面の透明度を上げることによって，切断した面から見た時に立体的に捉えることを容易にした．それによって，切断したところの ball と stick が浮き上がって見え，どのような構造をとっているか理解しやすくなった．
- 本研究を利用して，原子レベルの結晶成長の視覚化が可能となった．それによって，MSE の素過程である放出，拡散，吸着の物理現象が見てとれる．このように，MAYA によるアニメーションの作成によって，様々な物理現象が視覚化できるようになる．

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，ご協力を頂き，本研究を大成する事ができました．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．

関連図書

[1] 林田宏之，橋口智仁，konkon, 黒田あや子，本城なお,『AUTODESK MAYA』‘Mayaで始める 3DCG 制作の 基本‘，(株式会社ワークスコーポレーション,2006).

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

[3] Chris Pine, 西山伸,『初めてのプログラミング』,(株式会社オライリージャパン,2007).

[4] ジョージアルフケン,『ベクトルテンソルと行列』,(株式会社　島田製本所,1983).

[5] 由宇義珍,『はじめてのパワーデバイス』，(工業調査会,2006)

[6] 寺井健太,『SiC の表面エネルギーの第一原理計算』，(卒業論文，2008)