NISTIR 7234

Implementation of Simulation Program for Modeling the Effective Resistivity of Nanometer Scale Film and Line Interconnects

A. Emre Yarimbiyik
Harry A. Schafft
Richard A. Allen
Mona E. Zaghloul
David L. Blackburn

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 7234

Implementation of Simulation Program for Modeling the Effective Resistivity of Nanometer Scale Film and Line Interconnects

A. Emre Yarimbiyik
Mona E. Zaghloul
The Department of Electric and Computer Engineering,
The George Washington University
Washington, DC 20052

Harry A. Schafft
Richard A. Allen
David L. Blackburn
Semiconductor Electronics Division
Electronics and Electrical Engineering Laboratory

February 2006

U.S. DEPARTMENT OF COMMERCE
Carlos M. Gutierrez, Secretary
TECHNOLOGY ADMINISTRATION
Under Secretary of Commerce for Technology, William Jeffrey (acting as)
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
William Jeffrey, Director

Implementation of Simulation Program for Modeling the Effective Resistivity of Nanometer Scale Film and Line Interconnects

A. Emre Yarimbiyik[1, 2], Harry A. Schafft[2], Richard A. Allen[2], Mona E. Zaghloul[1], and David L. Blackburn[2]

1- The Department of Electric and Computer Engineering,
The George Washington University, Washington, DC 20052
2- Semiconductor Electronics Division,
National Institute of Standards and Technology, Gaithersburg, MD 20899

Abstract

A computer program that simulates the impact of the size effect on the effective resistivity of line and film conductors is described. Flowcharts and the program code are provided as appendices.

Introduction

A versatile program, with its flow chart and code, are provided to permit the reader to simulate how the effective resistivity of a metal conductor increases as the conductor cross sectional dimensions approach and become smaller than the bulk mean free path, λ_{max}, of the conduction electrons (size effect). For copper λ_{max} = 39 nm at room temperature. As integrated circuits (IC) are continuing to be designed with ever small conductor dimensions, the size effect on IC operating speed is a consideration of growing importance. (ITRS 2004 suggests dimensions smaller than 45 nm beyond 2009.) Hence, it is important to understand how grain size and the scattering characteristics of line and grain boundaries can increase the effective resistivity of lines and the R-C delays associated with the circuit wiring.

The program can model how scattering from surfaces and grain boundaries impact the effective resistivity of a film or line conductor when these scattering mechanisms act separately or simultaneously. The grain size, which may or may not be influenced by the dimensions of the conductor, is an important adjustable parameter in the program. The probability of an electron being scattered elastically from the surface can be made to be different for each surface of the conductor. This version of the program permits only one surface of a line to be different. Temperature effects are modeled by changing the value for the mean free path of the conduction electrons in the metal of interest. The application of this program was reported in another publication [1], where it was used to explore size effects in copper metallizations. While this program was designed for studying size effects in copper, it can be used equally well for other metallizations, such as aluminum and silver.

Summary of the Simulation

The code is developed for rectangular prism-shaped structures. Grain boundaries are assumed to be planes perpendicular to the sidewalls [3]. These planes are separated by a distance made equal to the estimated mean size of the grains in the metal. The simulation uses Sommerfeld's conductivity theory as expressed in eq. 1 to calculate the resistivity.

$$\sigma = \frac{N_f e^2 \tau}{m_{eff}} \quad , \tag{1}$$

where N_f is the number of electrons with energy near the Fermi energy per unit volume, e is the unit charge, τ is the relaxation time, and m_{eff} is the effective mass of an electron in the lattice. To calculate the simulated resistivity, all that is needed is to calculate the average relaxation time for the electrons in the conductor. This is what the program is designed to do. Values for N_f and the effective mass of the electron for a given metal exist in the literature and may be found in solid state physics books such as by Kittel [4].

To calculate the average relaxation time for the electrons in the conductor, the program simulates the movement of an electron in steps as it moves from one inelastic scattering event to the next. After each inelastic scattering event, the motion of the electron is modeled as moving at the Fermi speed, v_F, along a straight line that is directed in a randomly selected direction in the metal. If the electron is not scattered inelastically by a surface or grain boundary after it has traveled a distance equal to the bulk mean free path, λ_{max}, it is assumed to be inelastically scattered. The relaxation time for the electron during this simulated step in its motion in the metal is set equal to its maximum value, $\tau_{max} = \lambda_{max} / v_F$. If the electron is inelastically scattered before this time, it is assigned a value for the relaxation time that is equal to the product of τ_{max} and the fraction of λ_{max} that the electron has traveled during this step. The program calculates a value for the average of the relaxation times for each of N individual carriers (electrons) over M steps for a given geometry and grain size. Two parameters are used to characterize the scattering: p is the probability of elastic (specular) scattering from a surface and g is the probability of inelastic scattering by a grain boundary. Each of the N simulation electrons begins its first step by placing it in a randomly selected point in a grain and giving it a speed v_F in a randomly selected direction.

Setting the Parameters of the Program

The following parameters are initialized in the start of the main function.

MFP: Mean free path of electrons in metal. It is temperature dependent. This dependence is discussed in reference 1.
MAXTREL: Maximum relaxation time for electrons in the pure, bulk form of the metal. The bulk resistivity of copper for different temperatures is listed in reference 5. The user should calculate MAXTREL for copper using this information and equation 1.

pElastic: Surface scattering parameter, *p,* for three sides of the rectangular prism-shaped structure. Default value is 0.1. pElastic can take values between 0 and 1.

pElasticFourthSide: *p* value for the top side of the line. Default value is 0.1. pElasticFourthSide can take values between 0 and 1.

pGBSct: Grain boundary scattering parameter, *g*. Default value is 0.7. pGBSct can take values between 0 and 1.

Meff: Effective mass of electrons in the metal. It can be entered in terms of "M," the free electron mass, which is a constant in the program.

N_fermi: Number of electrons nearby the Fermi energy per volume. Note that this value is essentially temperature independent. For copper, it is equal to 8.47×10^{28} [4].

N_simel: Number of simulation electrons. Default value is 500.

N_step: Number of steps taken by each electron. Default value is 2000.

N_simrange: Number of different geometries that are to be simulated in one run. Default value is 30.

The parameters that define the dimensions are located in the "for" loop in the main function.

yvalue and **zvalue**: Width and height dimensions (note that all the lengths are in nanometers in the code) of the conductor line. For thin films, assigning 1000 times the "yvalue" to "zvalue" is a reasonably good approximation of a thin film. As the default, "yvalue" is 10 nm and is incremented by 10 nm in each loop. The number of loops is determined by "N_simrange."

grainSize: Average size of the grains. Grain size has crucial importance in taking the grain boundary scattering into account. Following typical usage reported in the literature, the default value for "grainSize" is the thickness of a film or the smaller cross sectional dimension of a line. However, to obtain the most accurate simulation results, use the value for the average size of the grains in the metallization determined from direct measurements.

REFERENCES

[1] Yarimbiyik, A.E., Schafft, H.A., Allen, R.A., Zaghloul, M.E., Blackburn, D.L. Modeling and Simulation of Resistivity of Nanometer Scale Copper, In press - Microelectronics Reliability 2006.
[2] Barnat, E.V., Nagakura, D., Wang, P-l, Lu, T-M. Real Time Resistivity Measurements During Sputter Deposition of Ultrathin Copper Films, JAP 2002;91:3:1667-72.
[3] Mayadas, A.F., Shatzkes, M. Electrical Resistivity Model for Polycrystalline Films: the Case of Arbitrary Reflection at External Surfaces, Phys. Rev. B 1970;1:4:1382-9.
[4] Kittel, C., Introduction to Solid State Physics. New York; John Wiley & Sons, Inc., 1996; p.150.
[5] Schuster, C.E., Vangel, M.G., Schafft, H.A. Improved Estimation of the Resistivity of Pure Copper and Electrical Determination of Thin Copper Film Dimensions, Microelectronics Reliability, 2001;41:239-52.

APPENDIX 1. Program Flowcharts

The following figures constitute the flowcharts of the Power Point file: FC_Cu.ppt.

Figure 1: Flowchart of the main routine.

Figure 2: Flowchart of the “Simulate Electron” function and a part of the main routine.

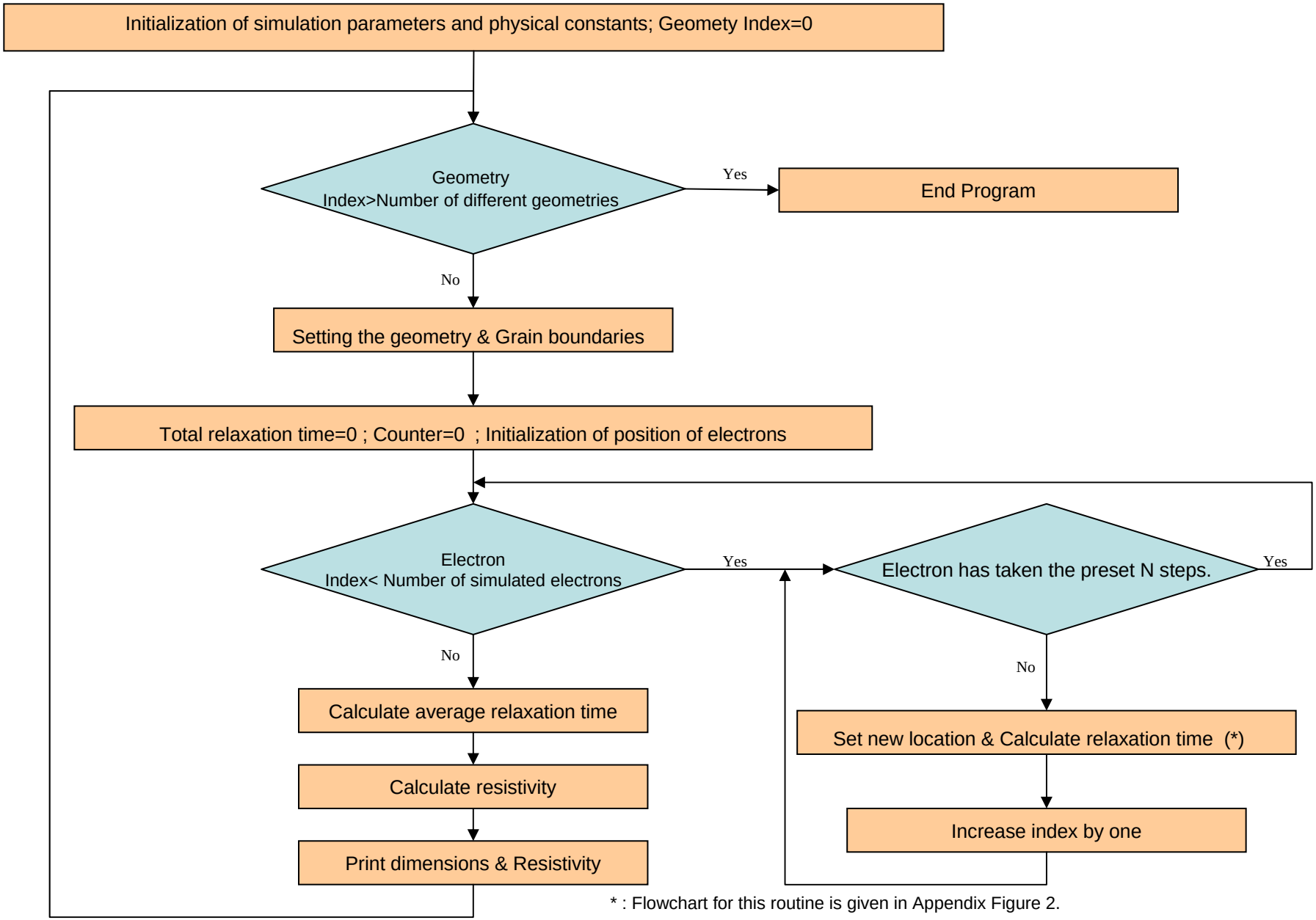
Figure 3: Flowchart of the elastic scattering routine.

Figure 4: Flowchart of the grain boundary check routine.

APPENDIX 2. Program Code

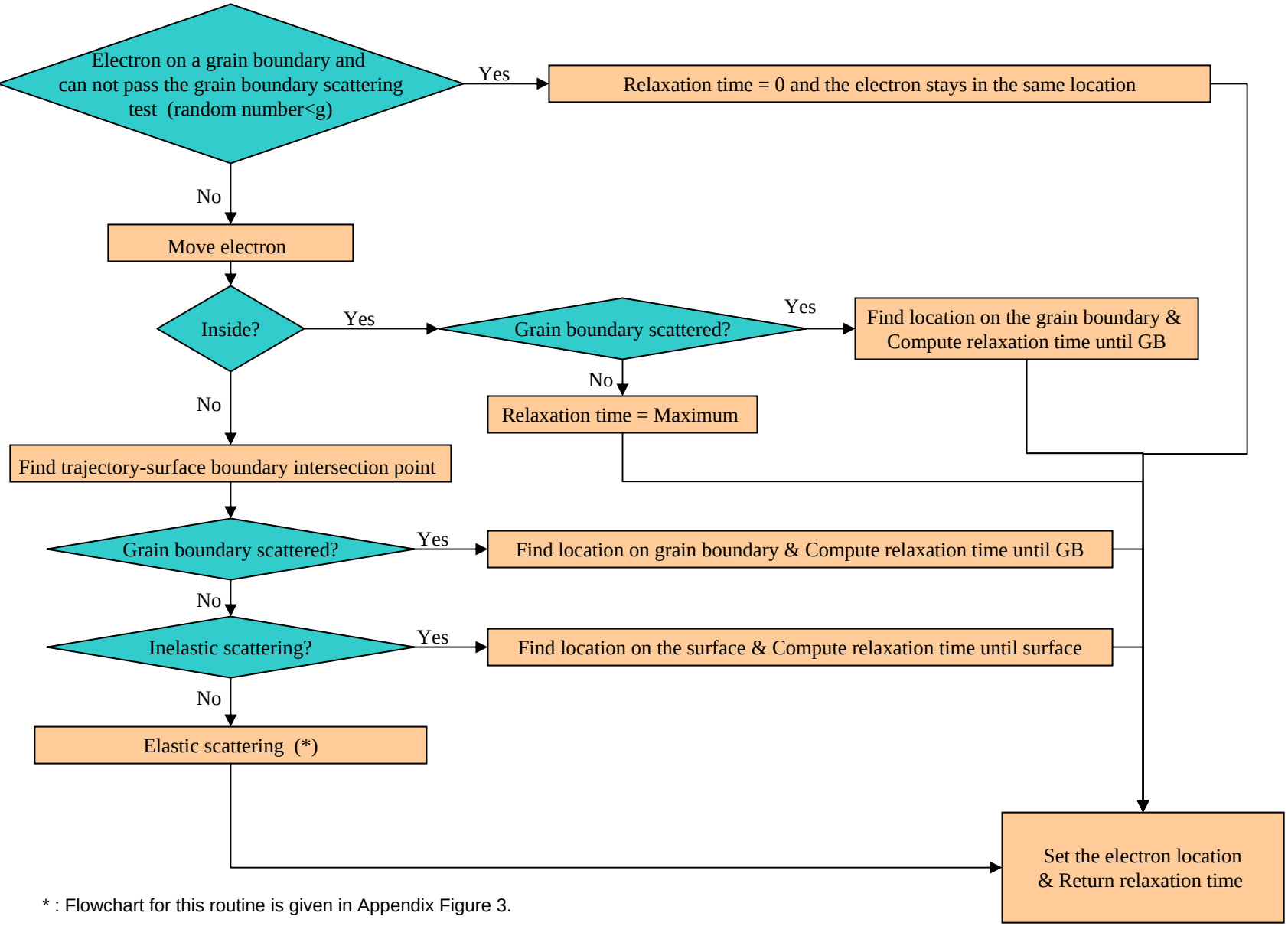
The code is given in a .java file, titled Copper.java.

Appendix 1 - Figure 1: Flowchart of the main routine



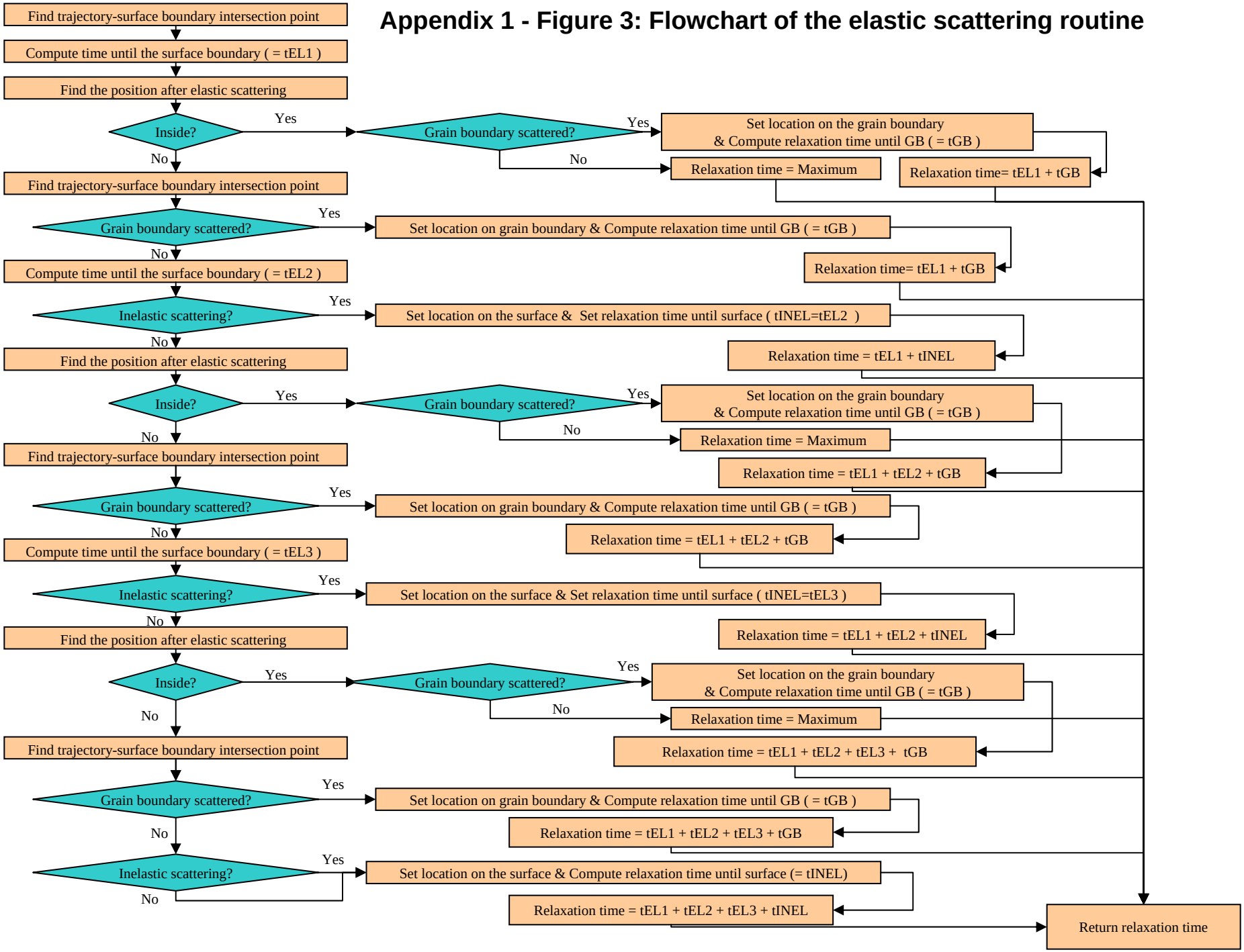
* : Flowchart for this routine is given in Appendix Figure 2.

Appendix 1- Figure 2: Flowchart of the "Simulate Electron" function and a part of the main routine

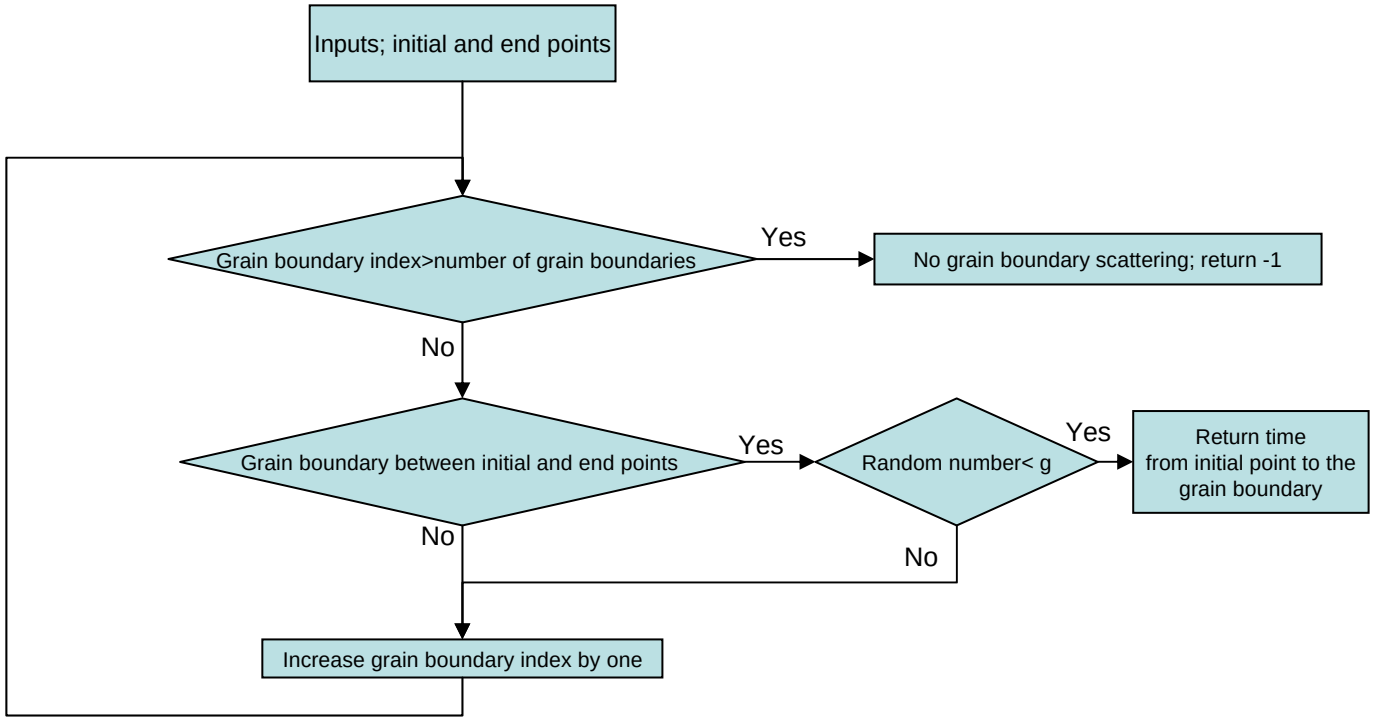


* : Flowchart for this routine is given in Appendix Figure 3.

Appendix 1 - Figure 3: Flowchart of the elastic scattering routine



Appendix 1 - Figure 4: Flowchart of the grain boundary check routine



The locations of the grain boundaries are stored in an array in the simulation. When the function that checks for the grain boundary scattering is called, the elements of the array are checked to be between the x coordinates of the initial and end points. If this condition is satisfied and the generated random number is bigger than the g parameter, then the electron is scattered from the grain boundary and the loop breaks. If this double check yields a positive result for none of the array elements, then there is no grain boundary scattering. The “grain boundary scattering check” function returns –1 for no grain boundary scattering. If there is grain boundary scattering, the electron’s position is set to the location that the electron’s trajectory intersects with the grain boundary. In this case the return value is the time that passes for the electron to travel from its initial position to the grain boundary

APPENDIX 2 - JAVA Source Code

This code was written using the jGRASP editor and has successfully been compiled using the Java J2SE v 1.4.2_08 SDK source code. A soft copy of the code is available from the authors.

```
/* Simulation program for size effects (surface scattering and grain boundary scattering)
   in resistivity of metal (Cu, Al, etc) films amd line interconnects;
   written by Arif Emre Yarimbiyik -- Version 1.0                                        */

import java.util.*;
import java.lang.Math;

public class Copper
{

    public final double convertmnm = 10e9;
    public final static double Ex = 4.0e10 ;
              // electric field in -x direction;  Note: Not used in this version !!!
    public final static double MFP=39;
    public final static double MAXTREL=3.1681716e-14;

    //  public final static double MFP=28.86;
    //  public final static double MAXTREL = 2.268e-14; //_ 125C

    public final static double PI=3.1415926;


    public static void main(String[]args)
    {
        final int N_simel = 500, N_simrange = 30;
        final double N_fermi = 8.47e28; // number of electrons nearby the Fermi energy per volume
        final double simulation_scale = N_simel/N_fermi;
        final double q = 1.6e-19;
        final double M = 9.035e-31;    // C , kg
        final double Meff = M * 1.3;
        final double N_step = 2000;
        final double startBox = N_step * MFP;
        final double xvalue = 2*startBox     ;
        final double pElastic=0.1;
        final double pElasticFourthSide=0.1;
        final double pGBSct=0.7;   //1.0;    //0.7;
        final double gStdevRatio=0.0;  // Stdev in grain size is not active currently
        double yvalue=0.0 , zvalue=0.0;
        double grainSize=0.0;
        double grainSizeStdev=0.0;
        double trel=0.0 , totalTrel=0.0 , totalStep=0.0;
        double conductivity=0.0,resistivity=0.0;
        double dummy=0.0;
        int noway;
        boolean onGB = false; // on a grain boundary ?

        for (int i=0 ; i<N_simrange ; i++)
        {
            totalTrel = 0.0;
            totalStep = 0L;
            yvalue = 10.0 + (double)i*10.0;
            zvalue = 1000*yvalue;    // for thin film
            grainSize=yvalue;        // MS assumption
            grainSizeStdev=gStdevRatio*grainSize;  // Not active !!
            Wire simwire = new
            Wire(xvalue,yvalue,zvalue,N_simel,pElastic,pElasticFourthSide,pGBSct,grainSize,grainSizeStdev);
            totalTrel = 0.0;
            for(int eIndex=0 ; eIndex<N_simel ; eIndex++)
            {
                for(int qwe=0; qwe<N_step ; qwe++)
                {
                    if ( !simwire.eConfig[eIndex].isOnGB(simwire.grainBnds) ||
                            ( simwire.eConfig[eIndex].isOnGB(simwire.grainBnds) && Math.random()>=pGBSct )     )
                    {
                        dummy=simwire.simulateElectron(eIndex) ;
                        totalTrel = totalTrel + dummy ;
                        noway = simwire.getEPosition(eIndex).unallowedDirection(yvalue,zvalue);
                        simwire.getEPosition(eIndex).moveInside(0.00000001,noway);
                    }
                    else totalTrel=totalTrel + 0.0;
```

```
                }
            }
            trel = totalTrel / (double) (N_step*N_simel) ;
            conductivity = calcConductivity(N_fermi,q,Meff,trel);
            resistivity = 1.0 / conductivity;
            System.out.println(/*"@@@@@@@@@@  "+*/ /*yvalue + "  " + trel + " st" +  "   "+*/resistivity);
        }
    }

    public static double smaller(double a,double b)
    {
        double small=a;
        if(b<a) small=b;
        return small;
    }

    public static double calcConductivity(double num,double q,double Meff,double tRelax)
    {
        return (num*q*q*tRelax) / Meff;
    }

}

////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////

class Wire
{
private double xBound;
private double yBound;
private double zBound;
private int NN;
private double probElas;
private double probElasFourth;
private double probGrainbndScattering;
private double grainSize;    // right now strict ..
private double grainSizeStdev;
public Coords[] eConfig;
public Coords[] grainBnds;

Wire()
{
    double xBound=1000;
    double yBound=30;
    double zBound=60;
    int NN=3;
    double probElas=0.1;
    double probGrainbndScattering=0.7;
    double grainSize=yBound;
    double grainSizeStdev=0.2*grainSize;
}

Wire(double xb,double yb,double zb,int num,double prob,double probFourth, double probgrainsct,
                                                                    double gSize, double gSizeStdev)
{
    xBound = xb;
    yBound = yb;
    zBound = zb;
    NN=num;
    probElas=prob;
    probElasFourth=probFourth;
    probGrainbndScattering=probgrainsct;
    eConfig = new Coords[NN];
    xBound=1000*Copper.MFP;
    int nGbnd=(int)(xBound/gSize)+20;
    grainBnds = new Coords[nGbnd];

    /* -------- Setting the grain boundaries -------------- */
    for(int a=0;a<nGbnd ;a++) grainBnds[a] = new Coords(0.0,0.0,0.0);
    int dumInd=0;
    for(int asd=0;asd<xBound ;asd++)
    {
        if(asd%gSize==0)
        {
            grainBnds[dumInd]=new Coords(asd,0.0,0.0);
            dumInd++;
        }
    }
    /* ------------------------------------------------- */
```

```
    /* -------- Setting the initial position of electrons ------------ */
    for(int i=0 ; i<eConfig.length ; i++)
        eConfig[i] = new Coords( (xBound/2.0)+(Math.random()*100000)%gSize  ,
                            (Math.random()*100000)%yBound  , (Math.random()*100000)% zBound  );
    /* ------------------------------------------------------------- */

}



public double simulateElectron(int eIndex)
{
    boolean inside,isGBS,fourthSide;
    int noway = 0;
    double amount = 0.0;
    double timeInelas = 0.0 , timeElas=0.0 , timeNosct=0.0 , timeGBSct=0.0 , trel=0.0;
    boolean grainbndOnTheWay=false;
    Coords newPosition = this.eConfig[eIndex];
    Coords initialPosition = new Coords(newPosition.getX(),newPosition.getY(),newPosition.getZ());

    newPosition.move(Copper.Ex);
    isGBS=isGrainbndScattered(probGrainbndScattering);

    inside = newPosition.checkifin(this.yBound,this.zBound);
    if(inside)
    {
        timeGBSct=newPosition.gbSCT(initialPosition,this.grainBnds,probGrainbndScattering);
        if(timeGBSct==-1.0)
        {
            timeNosct=Copper.MAXTREL;
            trel=timeNosct;
        }
        else
        {
            trel=timeGBSct;
        }
    }
    else
    {
        Coords newTempSurfacePosition = new Coords(newPosition.getX(),newPosition.getY(),newPosition.getZ());
        newTempSurfacePosition.findIntersect(initialPosition,this.yBound,this.zBound);
        timeGBSct=newTempSurfacePosition.gbSCT(initialPosition,this.grainBnds,probGrainbndScattering);
        if(timeGBSct==-1)
        {
            if(isElastic(probElas,probElasFourth,newTempSurfacePosition)==false)
            {
                timeInelas = newPosition.inelas(initialPosition,this.yBound,this.zBound);
                trel = timeInelas;
            }
            else
            {
                timeElas = newPosition.elas(initialPosition,this.xBound,this.yBound,this.zBound,probElas,
                                                        probElasFourth,this.grainBnds,probGrainbndScattering);
                trel = timeElas;
            }
        }
        else
        {
            newPosition.setXYZ(newTempSurfacePosition.x,newTempSurfacePosition.y,newTempSurfacePosition.z);
            trel=timeGBSct;
            // here newTempSurfacePosition is actually the position on the GB
        }
    }
    this.eConfig[eIndex] = newPosition;
    return trel;
}

public boolean isGrainbndScattered(double pGbSct)
{
    boolean gbSct=false;
    if(  Math.random() < pGbSct)   gbSct=true;
    return gbSct;
}

public boolean isElastic(double pel,double pelFourth,Coords intersec)
{
    boolean elasticSct=false;
    boolean sideFour=false;

    if (isBetween(intersec.getY(),-0.01,0.01)==true)  sideFour=true;
```

```
        if (sideFour)
        {
            if(  Math.random() < pelFourth)   elasticSct=true;
        }
        else
        {
            if(  Math.random() < pel)   elasticSct=true;
        }
        return elasticSct;
    }

    boolean isBetween(double q, double boundOne,double boundTwo)
    {
        boolean between = false;
        if ( ( q>boundOne && q<boundTwo ) || ( q<boundOne && q>boundTwo )    )
        {
            between = true;
        }
        return between;
        }

        public Coords getEPosition(int i)     {  return this.eConfig[i];    }

    }

////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////

    class Coords
    {

        double x;
        double y;
        double z;

        Coords()
        {
            double x = 1.0;
            double y = 1.0;
            double z = 1.0;
        }

        Coords(double xCoord,double yCoord,double zCoord)
        {
            x = xCoord;
            y = yCoord;
            z = zCoord;
        }
        public void setY(double q)   { y=q; }
        public void setXYZ(double p,double q,double r)   {  x=p; y=q ; z=r;    }
        public double getX()     {  return x;    }
        public double getY()     {  return y;    }
        public double getZ()     {  return z;    }
        public boolean isElastic(double pel,double pelFourth,Coords intersec)
        {
            boolean elasticSct=false;
            boolean sideFour=false;
            if (isBetween(intersec.getY(),-0.01,0.01)==true)        sideFour=true;
            if (sideFour)
            {
                if(  Math.random() < pelFourth)   elasticSct=true;
            }
            else
            {
                if(  Math.random() < pel)   elasticSct=true;
            }
            return elasticSct;
        }

        public double inelas(Coords initialPosition,double yBound,double zBound)
        {
            int noway = 0;
            double amount = yBound*0.0001;
            double timeInelas=0.0;
            noway = initialPosition.unallowedDirection(yBound,zBound);
            initialPosition.moveInside(amount,noway);
            this.findIntersect(initialPosition,yBound,zBound);
            noway = this.unallowedDirection(yBound,zBound);
            timeInelas = this.timePassed(Copper.MAXTREL,initialPosition);
            this.moveInside(amount,noway);
```

```
    return timeInelas;
}

public double elas(Coords initialPosition,double xBound,double yBound,double zBound,double
                probElas,double probElasFourth,Coords[] grainB,double probGBS)
{
    double timeElas=0.0 , timeInelas=0.0;
    double timeGBSct=0.0;
    int noway;
    Coords symmetric = new Coords(this.getX(),this.getY(),this.getZ());
    this.findIntersect(initialPosition,yBound,zBound);
    double t_elasfirst = this.timePassed(Copper.MAXTREL,initialPosition);
    double t_elassecond=0.0;
    initialPosition.setXYZ(this.getX(),this.getY(),this.getZ());
    noway = initialPosition.unallowedDirection(yBound,zBound);
    initialPosition.moveInside(0.00001,noway);
    this.findElastic(symmetric,yBound,zBound);
    boolean inside = this.checkifin(yBound,zBound);
    if(inside)
    {
        timeGBSct=this.gbSCT(initialPosition,grainB,probGBS);
        if(timeGBSct==-1.0)
        {
            timeElas = Copper.MAXTREL ;
            timeInelas = 0.0;
        }
        else
        {
            timeInelas=timeGBSct;
            timeElas=t_elasfirst;
        }
    }
    else
    {
    Coords newTempSurfacePosition = new Coords(this.getX(),this.getY(),this.getZ());
    newTempSurfacePosition.findIntersect(initialPosition,yBound,zBound);
    timeGBSct=newTempSurfacePosition.gbSCT(initialPosition,grainB,probGBS);
    if(timeGBSct==-1)
    {
        if(isElastic(probElas,probElasFourth,newTempSurfacePosition)==false)
        {
            timeInelas = this.inelas(initialPosition,yBound,zBound);
        timeElas=t_elasfirst;
        }
        else  // ................. 2nd(++) elas.....................
        {
            Coords symmetricTwo = new Coords(this.getX(),this.getY(),this.getZ());
            this.setXYZ( newTempSurfacePosition.getX(), newTempSurfacePosition.getY(),
            newTempSurfacePosition.getZ());
            noway = this.unallowedDirection(yBound,zBound);
            this.moveInside(0.00001,noway);
            t_elassecond = this.timePassed(Copper.MAXTREL,initialPosition);
            initialPosition.setXYZ(this.getX(),this.getY(),this.getZ());
            this.findElastic(symmetricTwo,yBound,zBound);
            inside = this.checkifin(yBound,zBound);
            if(inside)
            {
                timeGBSct=this.gbSCT(initialPosition,grainB,probGBS);
                if(timeGBSct==-1.0)
                {
                    timeElas = Copper.MAXTREL;
                    timeInelas = 0.0;
                }
                else
                {
                    timeInelas=timeGBSct;
                    timeElas=t_elasfirst + t_elassecond;
                }
                inside=this.checkifin(yBound,zBound);
            }
            else // 2nd elas out ..
            {
                Coords nTempSurfPosTwo = new Coords(this.getX(),this.getY(),this.getZ());
                Coords checky = new Coords(this.getX(),this.getY(),this.getZ());
                noway = initialPosition.unallowedDirection(yBound,zBound);
                initialPosition.moveInside(0.00001,noway);
                nTempSurfPosTwo.findIntersect(initialPosition,yBound,zBound);
                noway = nTempSurfPosTwo.unallowedDirection(yBound,zBound);
                nTempSurfPosTwo.moveInside(0.00001,noway);
                timeGBSct=nTempSurfPosTwo.gbSCT(initialPosition,grainB,probGBS);
```

```
                    if(timeGBSct!=-1)
                    {
                        this.setXYZ(nTempSurfPosTwo.x,nTempSurfPosTwo.y,nTempSurfPosTwo.z);
                        timeElas=t_elasfirst+t_elassecond;
                        timeInelas=timeGBSct;
                        // here newTempSurfacePosition is actually the position on the GB
                        noway = this.unallowedDirection(yBound,zBound);
                        this.moveInside(0.00001,noway);
                        inside=this.checkifin(yBound,zBound);
                    }
                    else
                    {
                    if(isElastic(probElas,probElasFourth,nTempSurfPosTwo)==false)
                    {
                        timeInelas = this.inelas(initialPosition,yBound,zBound);
                        noway = this.unallowedDirection(yBound,zBound);
                        this.moveInside(0.00001,noway);
                        timeElas=t_elasfirst+t_elassecond;
                        inside=this.checkifin(yBound,zBound);
                    }
                    else  // ................. 3rd elas.....................
                    {
                        double t_elasthird=0.0;
                        inside=initialPosition.checkifin(yBound,zBound);
//    3rd elas starts  ////////////////////////////////////////////////////////////////////////////////
                        Coords symmetricThree = new Coords(this.getX(),this.getY(),this.getZ());
                        this.findIntersect(initialPosition,yBound,zBound);
                        noway = this.unallowedDirection(yBound,zBound);
                        this.moveInside(0.00001,noway);
                        t_elasthird = this.timePassed(Copper.MAXTREL,initialPosition);
                        initialPosition.setXYZ(this.getX(),this.getY(),this.getZ());
                        this.findElastic(symmetricThree,yBound,zBound);
                        inside = this.checkifin(yBound,zBound);
                        if(inside)
                        {
                            timeGBSct=0.0;
                            timeGBSct=this.gbSCT(initialPosition,grainB,probGBS);
                            if(timeGBSct==-1.0)
                            {
                                timeElas = Copper.MAXTREL;
                                timeInelas = 0.0;
                            }
                            else
                            {
                                timeInelas=timeGBSct;
                                timeElas=t_elasfirst + t_elassecond +  t_elasthird;
                            }
                            inside=this.checkifin(yBound,zBound);
                        }
                        else // 3rd elas out ..
                        {
                            Coords nTSPThree = new Coords(this.getX(),this.getY(),this.getZ());
                            noway = initialPosition.unallowedDirection(yBound,zBound);
                            initialPosition.moveInside(0.00001,noway);
                            nTSPThree.findIntersect(initialPosition,yBound,zBound);
                            noway = nTSPThree.unallowedDirection(yBound,zBound);
                            nTSPThree.moveInside(0.00001,noway);
                            timeGBSct=nTSPThree.gbSCT(initialPosition,grainB,probGBS);
                            if(timeGBSct!=-1)
                            {
                                this.setXYZ(nTSPThree.x,nTSPThree.y,nTSPThree.z);
                                timeElas = t_elasfirst + t_elassecond + t_elasthird;
                                timeInelas=timeGBSct;
                                noway = this.unallowedDirection(yBound,zBound);
                                this.moveInside(0.00001,noway);
                                inside=this.checkifin(yBound,zBound);
                            }
                            else
                            {
                                if(isElastic(probElas,probElasFourth,nTSPThree)==false)
                                {
                                    timeInelas = this.inelas(initialPosition,yBound,zBound);
                                    noway = this.unallowedDirection(yBound,zBound);
                                    this.moveInside(0.00001,noway);
                                    timeElas=t_elasfirst+t_elassecond+t_elasthird;
                                    inside=this.checkifin(yBound,zBound);
                                }
                                else  // ................. !!! NO   4th(and ++) elas  ALLOWED !!! ......
                                {
                                    timeInelas = this.inelas(initialPosition,yBound,zBound);
```

```
                                    noway = this.unallowedDirection(yBound,zBound);
                                    this.moveInside(0.00001,noway);
                                    timeElas=t_elasfirst+t_elassecond+t_elasthird;
                                    inside=this.checkifin(yBound,zBound);
                                }
                            }
                        }
//    3rd elas  ends ////////////////////////////////////////////////////////////////////////////////////////
                    }
                }
            }
        }
        }
        else
        {
            this.setXYZ(newTempSurfacePosition.x,newTempSurfacePosition.y,newTempSurfacePosition.z);
            timeElas=t_elasfirst;
            timeInelas=timeGBSct;     // here newTempSurfacePosition is actually the position on the GB
            inside=this.checkifin(yBound,zBound);
        }
    }
    return timeElas+timeInelas;
    }

    public void findElastic(Coords sym , double yBound , double zBound)
    {
        Coords middle = new Coords();
        double eps=0.01;
        if(isBetween(this.z,zBound-eps,zBound+eps) )
        {
            middle.z = zBound;
            middle.y = sym.y;
        }
        if(isBetween(this.z,0.0-eps,0.0+eps))
        {
            middle.z = 0.0;
            middle.y = sym.y;
        }
        if(isBetween(this.y,yBound-eps,yBound+eps) )
        {
            middle.y = yBound;
            middle.z = sym.z;
        }
        if(isBetween(this.y,0.0-eps,0.0+eps) )
        {
            middle.y = 0.0;
            middle.z = sym.z;
        }
        this.y = 2.0 * middle.y - sym.y ;
        this.z = 2.0 * middle.z - sym.z ;
        this.x = sym.x ;
    }

    public double distanceFrom(Coords second)
    {
    //finds the distance between 2 points in 3-D space
        double dist;
        dist=Math.sqrt( Math.pow((this.x-second.x),2.0)+Math.pow((this.y-second.y),2.0)+
        Math.pow((this.z-second.z),2.0) );
        return dist;
    }

    public void move(double E_field)
    {
    // move sph.shell without direction restrictions
        Coords onsphere = new Coords();
        double theta,phi,convert = Copper.PI/180.0 ;
        theta = ( (Math.random()*100000) % 360) * convert;
        phi = ( (Math.random()*100000) % 180) * convert;
        onsphere.x = Copper.MFP * Math.sin(phi) * Math.cos(theta);
        onsphere.y = Copper.MFP * Math.sin(phi) * Math.sin(theta);
        onsphere.z = Copper.MFP * Math.cos(phi);
        this.x = this.x + onsphere.x;
        this.y = this.y + onsphere.y;
        this.z = this.z + onsphere.z;
    }

    public boolean checkifin(double yBound,double zBound)
    {
        boolean in = false;
```

```
    Coords eCheck = this;
    if ( eCheck.y >= 0.0  && eCheck.z >= 0.0  && eCheck.y <= yBound && eCheck.z <= zBound )
    {
        in = true;
    }
    return in;
}


int unallowedDirection(double yBound,double zBound)
{
// noway=1 no east noway=-1 no west
// noway=2 no north noway=-2 no south
// noway=0 all directions allowed
    int noway=0;
    double eps=0.0001;
    if  ( this.isBetween('z',zBound-eps,zBound+eps) )           noway = 1;
    else if( this.isBetween('z',0.0-eps,0.0+eps) )              noway =-1;
    else if( this.isBetween('y',yBound-eps,yBound+eps) )        noway = 2;
    else if( this.isBetween('y',0.0-eps,0.0+eps) )              noway =-2;
    return noway;
}

boolean isBetween(char side, double boundOne,double boundTwo)
{
    boolean between = false;
    double q = 0.0;
    if(side=='y') q = this.y;
    else if(side=='z') q= this.z;
    if ( ( q>boundOne && q<boundTwo ) || ( q<boundOne && q>boundTwo )    )
    {
        between = true;
    }
    return between;
}

boolean isBetween(double q, double boundOne,double boundTwo)
{
    boolean between = false;
    if ( ( q>boundOne && q<boundTwo ) || ( q<boundOne && q>boundTwo )    )
    {
        between = true;
    }
    return between;
}

public void moveInside(double amountin,int noway)
{
    if (noway==1)      this.z = this.z - amountin;
    else if(noway==-1) this.z = this.z + amountin;
    else if(noway== 2) this.y = this.y - amountin;
    else if(noway==-2) this.y = this.y + amountin;
}

double line3D(double c,double b1,double b2,double c1,double c2)
{
    /* Finds the needed coordinate of a point (of which one coordinate is known) on a 3D line

            Evaluates

                      y-y1
              z = -------- (z2-z1) + z1
                      y2-y1

           for    line3D(intersection.y,zone,ztwo,yone,ytwo);
//  z =  line3D(intersection.y,zone,ztwo,yone,ytwo);
//                  a          b1   b2   c1   c2
*/
    double bSeeked;
    bSeeked = ( (c-c1)/(c2-c1) ) * (b2-b1) + b1;
    return bSeeked;
}


public void findIntersect(Coords initPos,double yBound,double zBound)
{
/*  Assumption: Projection of electron's trajectory is a line in 3D
       Its projection on yz plane is a line that starts at (y1,z1)
       and ends at (y2,z2)
```

```
     y = a*z + b      (= z=(y-b)/a
                 y2-y1
     where a = -------  and b = y1-a*z1
                 z2-z1
    Our wire is in rectengular prism shape.
    Intersections of the trajectory with the lines
    y=0 , z=0 , y=maxY  and z=maxZ are found.
    Then, the intersection point which is actually on the boundaries of the
    rectengular prism is determined
    */

    double yone = initPos.y , zone = initPos.z , ytwo = this.y ,  ztwo = this.z;
    double xone = initPos.x , xtwo = this.x;
    double maxY = yBound , maxZ = zBound;
    double a,b;
    double solnUp= -1.0,solnDown=-1.0,solnLeft=-1.0,solnRight=-1.0;

//  z;y=maxY    z;y=0         y;z=0         y;z=maxZ

          a = (ytwo-yone) /  (ztwo-zone);
          b = yone-a*zone;

    if ( (ytwo-yone) != 0.0)
    {
           // z=(y-b)/a
        solnUp    = (maxY-b)/a;
        solnDown  =  -b/a;
    }

    if ( (ztwo-zone) != 0.0)
    {                       // y=a*z+b
        solnRight = a*maxZ+b;
        solnLeft  = b;
    }

    if( isBetween(solnUp,0.0,maxZ)  && isBetween(solnUp,zone,ztwo)  )
    {
        this.z = solnUp;
        this.y = line3D(this.z,yone,ytwo,zone,ztwo);
        this.x = line3D(this.z,xone,xtwo,zone,ztwo);
    }
    else if(  isBetween(solnDown,0.0,maxZ)  && isBetween(solnDown,zone,ztwo)  )
    {
        this.z = solnDown;
        this.y = line3D(this.z,yone,ytwo,zone,ztwo);
        this.x = line3D(this.z,xone,xtwo,zone,ztwo);
    }
    else if( isBetween(solnRight,0.0,maxY)  && isBetween(solnRight,yone,ytwo)  )
    {
        this.y = solnRight;
        this.z = line3D(this.y,zone,ztwo,yone,ytwo);
        this.x = line3D(this.y,xone,xtwo,yone,ytwo);
    }
    else if( isBetween(solnLeft,0.0,maxY)  && isBetween(solnLeft,yone,ytwo)  )
    {
        this.y = solnLeft;
        this.z = line3D(this.y,zone,ztwo,yone,ytwo);
        this.x = line3D(this.y,xone,xtwo,yone,ytwo);
    }
    if ( Math.abs(this.y)<0.0000001 ) this.y = 0.0;
    if ( Math.abs(this.z)<0.0000001 ) this.z = 0.0;
}

public double timePassed(double tmax,Coords second)
{
    // finds the time passed till an e's path intersects with the boundary
    //   first 1 ; last 2 ; intersect 3
    // computes the relaxation time for the electrons
    // that are inelastically scattered from the surface
    double ratio = 1.0;
    double way = 0.0;
    way = this.distanceFrom(second);
    ratio = way / Copper.MFP;
    return tmax * ratio;
}

public boolean isGrainbndScattered(double pGbSct)
{
    boolean gbSct=false;
    if(  Math.random() < pGbSct)   gbSct=true;
```

```
    return gbSct;
}

public boolean checkGrain(Coords initPos,Coords[] grainB)
{
    boolean gInBetween=false;
    int i;
    for(i=0;i<grainB.length;i++)
    {
        if (  ( this.x > grainB[i].getX() && initPos.getX() < grainB[i].getX() )  ||
                    ( this.x < grainB[i].getX() && initPos.getX() > grainB[i].getX() )     )
        {
            gInBetween=true;
            break;
        }
    }
    return gInBetween;
}

public double grainSct(Coords initPos,Coords[] grainB)
{
    // first find the intersection(the point which the carrier is scattered on the grain boundary)
    // then compute time passed
    double timeTillSct,tmax;
    int i;
    double xCoordIntersect,yCoordIntersect,zCoordIntersect;
    Coords pointOnGb=new Coords(0.0,0.0,0.0);
    for(i=0;i<grainB.length;i++)
    {
        if (  ( this.x > grainB[i].getX() && initPos.getX() < grainB[i].getX() )  ||
                    ( this.x < grainB[i].getX() && initPos.getX() > grainB[i].getX() )     )
        {
            xCoordIntersect = grainB[i].x;
            yCoordIntersect = this.y + ((xCoordIntersect-this.x) / ((this.x-initPos.x) /
                                                                    this.y-initPos.y)));
            zCoordIntersect = this.z + ((xCoordIntersect-this.x) / ((this.x-initPos.x) /
                                                                      (this.z-initPos.z)));
            pointOnGb = new Coords(xCoordIntersect,yCoordIntersect,zCoordIntersect);
            break;
        }
    }

    tmax=Copper.MAXTREL;
    timeTillSct = Math.abs(initPos.timePassed(tmax,pointOnGb));
    this.x=pointOnGb.x;
    this.y=pointOnGb.y;
    this.z=pointOnGb.z;
    return timeTillSct;
    }

public double gbSCT(Coords initPos,Coords[] grainB,double pGBS)
{
    double timeTillSct,tmax;
    int i;
    double xCoordIntersect,yCoordIntersect,zCoordIntersect;
    boolean gbScattered=false;
    int randDirection=0;
    Coords pointOnGb=new Coords(0.0,0.0,0.0);
    for(i=0;i<grainB.length;i++)
    {
        if (  ( this.x > grainB[i].getX() && initPos.getX() < grainB[i].getX() )  ||
                    ( this.x < grainB[i].getX() && initPos.getX() > grainB[i].getX() )         )
        (
            if(isGrainbndScattered(pGBS)==true)
            {
                xCoordIntersect = grainB[i].x;
                yCoordIntersect = this.y + ((xCoordIntersect-this.x) / ((this.x-initPos.x) /
                                                                        (this.y-initPos.y)));
                zCoordIntersect = this.z + ((xCoordIntersect-this.x) / ((this.x-initPos.x) /
                                                                        (this.z-initPos.z)));
                pointOnGb = new Coords(xCoordIntersect,yCoordIntersect,zCoordIntersect);
                gbScattered=true;
                break;
            }
        }
    }

    if(gbScattered==false)    return -1.0;
    else
    {
```

```
            tmax=Copper.MAXTREL;
            timeTillSct = Math.abs(initPos.timePassed(tmax,pointOnGb));
            this.x=pointOnGb.x;
            this.y=pointOnGb.y;
            this.z=pointOnGb.z;
            return timeTillSct;
        }
    }

    public boolean isOnGB(Coords[] gbs)
    {
        boolean onaGB=false;
        for(int i=0; i<gbs.length ;i++)
        {
            if( this.x == gbs[i].getX() )
            {
                onaGB=true;
                break;
            }
        }
        return onaGB;
    }

         public void tooString()
         {
             System.out.println( " ( " + this.x + " , " + this.y + " , " + this.z + " ) " );
         }

}
```