PHYS 370
Computational Physics

Jacob William Connelly
David Jedynak
Nick Zahler

March 07, 2018

1 Abstract

The semiconductor device is where electrical engineering meets quantum physics. By understanding the crystalline lattice structure of molecules and how charge carriers move through them, the ability to manipulate them was derived. 'Doping', the deliberate process of adding impurities into a structure, is the base theory behind the modern semiconductor. By placing two (differently) doped structures next to one another, a junction is created; this meeting of doped crystalline structures is the science behind all modern day electronics, including transistors and integrated circuits. Our experiment will seek to simulate with 3D visualizations based on C code the most basic of the semiconductor structures, the diode; a device that, when doped, only allows current to flow unidirectionally; this paper is the documentation of that experiment.

2 Introduction

To work through this problem we first needed to understand, in depth, the inner workings of the diode and how electricity moves through it, and why. This process was made relatively easy, since all group members have a background in electrical engineering. The trick was in figuring out how to take our knowledge of the physical chemistry of a diode, the understanding of its electrical properties, and translate that into a computer simulation. The diode simulation takes the form of a plane with various fields placed along its length. For easy visualization, the different fields are denoted by lines that stretch across the plane. When the program runs with a voltage source applied in the 'proper' orientation, current will flow across the diode. However, as would happen in reality, if the voltage source is 'backwards' (or, in effect, the diode turned the other direction), current flow stops.

3 Theory

Semiconductors are unique in the electrical world in that they are able to both conduct and repel electric current. They are made of material that has an electrical conductivity between strong electric conductors, such as gold or copper, and insulators, such as rubber or plastic. The most commonly used element for semiconductor devices is silicon, a metalloid in the periodic table. Pure silicon has no free electrons, which is why the element is doped to give it enhanced electroconductive properties. The doping process includes introducing impurities into the silicone crystal, as can be seen in Figure 1.

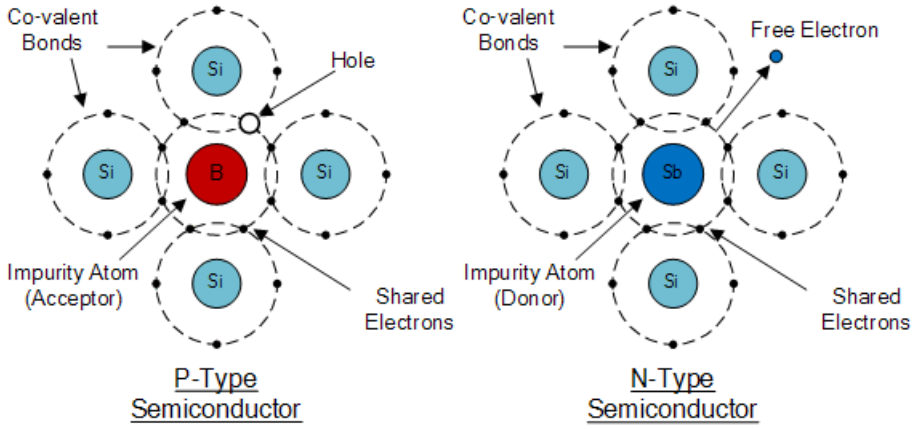

Figure 1: Diagram of doping process

As noted in the figure above, N-type doping adds an impurity to the silicone structure that introduces free electrons, while P-type introduces vacant positions, or 'holes', for free electrons to fill.

A diode is the physical object that puts this doping into practice at its most basic level. At the junction where P-type and N-type meet is where things get interesting. The extra electrons from the N-type transition over to the P-side, which gives the P-side a slightly negative charge. Simultaneously, since the electrons have moved out of the N-type region, that area gets a slightly positive charge. The electric field that results from this is known as the depletion region, which can be seen below.

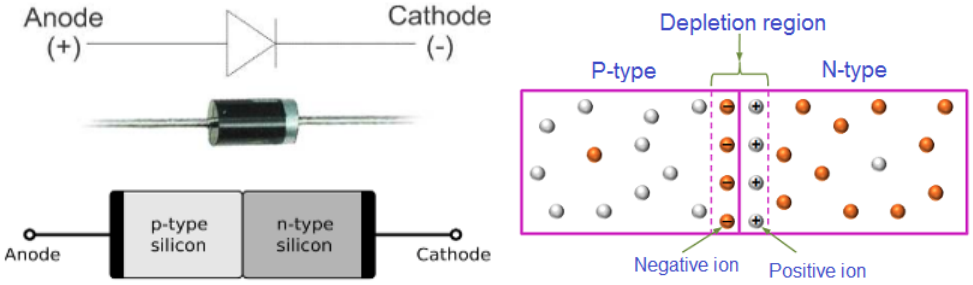

Figure 2: Physical diode and representations (left), Depletion region (right)

The depletion region is the basis of the barrier that stops electron flow. Applying current to the diode "backwards" results in what is known as reverse biasing, in which the depletion zone widens as power source attracts both electrons and holes. However, when connected the other way, and if an electron has enough energy, it can overcome the barrier potential and cross the depletion region. Negative electrons are pushed away by negative terminal, and "fall into the holes", depleted of energy. The electrons behind it cross the barrier and occupy the next line of holes, and so on, until the holes are full, and electricity passes through the diode. This result is the one-way flow of electricity.

In an ideal world with perfect conditions, the graph of Voltage vs Current (V vs I) should be zero at all points from negative infinity to 0V, then instantaneously spike along the y-axis at V>=0. In effect, the diode would be treated as nothing but a wire. In reality there are physical limitations on the chemical structure of silicone, which is why we need to reach a certain voltage potential before the barrier potential can be crossed. For silicone diodes, this happens at right around 0.7V. In the other direction, if a strong enough negative voltage is applied to the diode, it will eventually overpower the depletion zone and current will flow. This phenomena is known as 'breakdown', but only happens at voltages stronger than -50V, and will not be simulated in our experiment. The voltage vs current graphs of ideal vs real diodes is shown below.

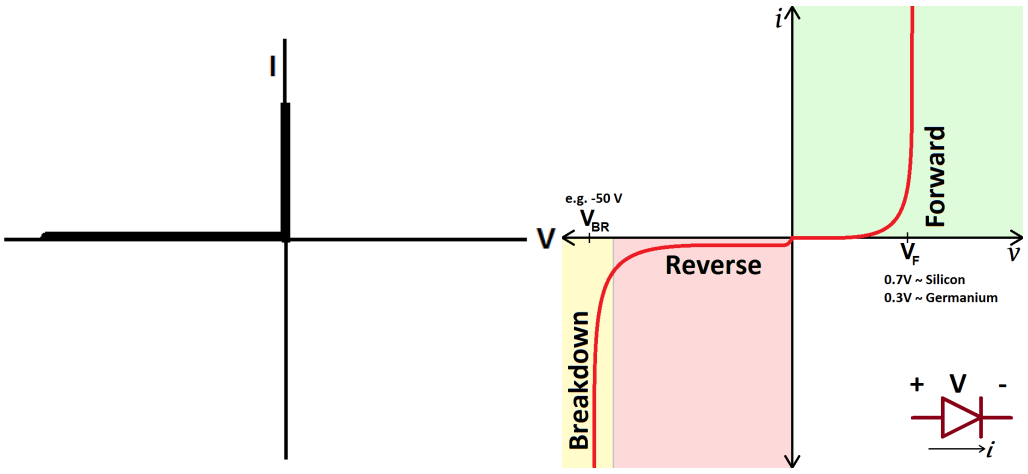

Figure 3: Ideal V vs I graph (left), Real V vs I graph (right)

4 Simulation Design

The general characteristics the Diode simulation is trying to model are the ability of diodes to allow current flow in one direction, the non linear relation of diode voltage and diode current, reverse bias current breakdown(i.e. exposed to a large voltage, the diode will allow reverse current), and the effect of changing the forward and reverse gap voltages. To create these behaviors, a voltage source, diode, and a resistance are needed.

The existing simulation(MDTherm.c) from PHYS370 has most of the dynamics necessary to simulate a diode. By adding charges to the particles, they can behave similar to electrons in a circuit. To simulate the voltage source, an area in the simulation box with a vertical, horizontal, length, and width is defined. If a particle is in this "voltage" box, a force in the vertical direction is applied to move the particles and simulate electrical current. The diode is composed of 2 of these force fields(forward gap and reverse gap) that are similar to the voltage source, but the forward gap only acts on particles moving with positive vertical velocity, and the downward gap acts on those with negative vertical velocity.

4.1 Theory-Mathematics

We need to be able to set voltages using force. Using the equations below a relation between force and voltage can be derived.

Where E Electric Field, V Voltage, Q particle charge, dl Field Length

$V = \vec{E} * dl$ $\vec{E} = \frac{\vec{N}}{Q}$

Derived relation between Voltage and Force:

$\vec{N} = \frac{V*Q}{dl}$

Similarly this can done for resistance:

4.1.1 Requirements

Inputs

1. Voltage Source (Voltage)
2. Particle Charge
3. Forward Diode Gap (Voltage)
4. Reverse Diode Gap (Voltage)
5. Voltage Sweep v start, v end, increment voltage, averaging time

Outputs

1. .dat file with the VI Curve for analysis

2. Real time current graph

3. graphics lines on the simulation to notate circuit components

4.2 Code Explained

Subroutines This subroutine calculates the average current by summing the voltages, dividing by box length (L) and multiplying by particle charge.

```
// functions defined for midterm
double getCurrent(double v[N][D]){
        double sumV = 0;
        for(int n = 0;n<N;n++){
                sumV += v[n][1];
                }
        return iq*sumV/L;
}
```

This subroutine calculates the forces for the different components based on the input voltages, resistances. It also sets the charge of the particles. This makes it easier to change parameters quickly while simulating

```
void setVoltageSource(){
  //calculate forces for the given voltages
  v_f = iq*vs_v/volt_len;                //calculate the field from the set voltage
  ud_f = iq*ud_v/diode_len_p;                //calculate the field from the set upper diode voltag
  dd_f = iq*dd_v/diode_len_n;                //calculate the field from the lower diode votlage ga
  for (int n=0; n<N; n++){
    for (int d=0; d<D; d++){
    q[n] = iq;                                 //set initial charge for each electron
    }}

}
```

This routine sweeps the voltage from start to end with an increment, outputs a file with the voltage and current information to look at diode VI characteristics.

```
void VI_Curve(){
  FILE *res;
  char IsoName[100];
  double sumI = 0;

  sprintf(IsoName,"VI_Curve_Q_%f_N_%i_ud_v_%f_dd_v_%f.dat",iq,N,ud_v,dd_v);//put info in the filer
  res=fopen(IsoName,"w");
  //sweep the voltage
  if(v_incr < 0 ){
          for (double ivolt = v_start;ivolt>v_end;ivolt = ivolt + v_incr){
                //set the voltage
                vs_v = ivolt;
                setVoltageSource();
                //loop to waste some time to get the current to settle
                for(int t = 0;t<v_sweep_delta;t++)
                        {
                        iterate(x,v,dt);
                                Events(1);
                        DrawGraphs();
                        sumI += current_var;
                        }
                //save the current and voltage data
                fprintf(res,"%e %f\n",sumI/v_sweep_delta,vs_v);
```

```
        }
  }
    else{
        for (double ivolt = v_start;ivolt<v_end;ivolt = ivolt + v_incr){
            //set the voltage
            vs_v = ivolt;
            setVoltageSource();
            //loop to waste some time to get the current to settle
            for(int t = 0;t<v_sweep_delta;t++)
                    {
                    iterate(x,v,dt);
                            Events(1);
                    DrawGraphs();
                    sumI += current_var;
                    }
            //save the current and voltage data
            fprintf(res,"%e %f\n",sumI/v_sweep_delta,vs_v);
        }
  }

  fclose(res);
}
```

Code added to draw() subroutine to draw lines where the voltage source, diode, and resistor fields exist.

```
//voltage source lines
  mydrawline(2,0,scalefac*(L-volt_pos+volt_len/2),size,scalefac*(L-volt_pos+volt_len/2));
  mydrawline(2,0,scalefac*(L-volt_pos-volt_len/2),size,scalefac*(L-volt_pos-volt_len/2));

//diode field 1 lines
  mydrawline(3,0,scalefac*(L-diode_pos_p + diode_len_p/2),size,scalefac*(L-diode_pos_p + diode_len
  mydrawline(3,0,scalefac*(L-diode_pos_p - diode_len_p/2),size,scalefac*(L-diode_pos_p - diode_len

//diode field 2 lines
 // mydrawline(4,0,100,size,100);
  mydrawline(4,0,scalefac*(L-diode_pos_n - diode_len_n/2),size,scalefac*(L-diode_pos_n - diode_len

//resistor field lines
  mydrawline(6,0,scalefac*(L-res_pos+res_len/2),size,scalefac*(L-res_pos+res_len/2));
  mydrawline(6,0,scalefac*(L-res_pos-res_len/2),size,scalefac*(L-res_pos-res_len/2));
```

Code added to iterate routine This code iterates through each particle and determines a force to apply to each particle depending on their position and velocity. This simulates the voltage source, diode, and a resistor(the resistor is untested).

```
    for (int n=0;n<N;n++)
      for (int d=0;d<D;d++){
        //begin conditions for diode, resistor, and voltage source regions
        v[n][d]+=(ff[n][d])/mass[n]*dt;// integrate to get velocity

        if (d == 1){// if the dimension is in the Y dimension... vertical
        //check to see if the particle is in diode p, diode n, voltage, resistor regions, then app
                if ((x[n][d] < (diode_pos_p+diode_len_p/2)) && (x[n][d] > (diode_pos_p-diode_len_p
                        field_force = ud_f - current_var;//simulate the forward diode band gap
                        }
                else if((x[n][d] < (diode_pos_n+diode_len_n/2)) && (x[n][d] > (diode_pos_n-diode_l
                        field_force = dd_f + current_var;//simulate the reverse diode band gap
                        }
```

```
                else if((x[n][d] < (volt_pos + volt_len/2)) && (x[n][d] > (volt_pos - volt_len/2))
                        field_force = v_f;//simualate voltage source
                        }
                else if((x[n][d] > (res_pos - res_len/2)) && (x[n][d] < (res_pos + res_len/2)) &&
                        field_force = -v[n][d]*q[n]*q[n]*resistance/(res_len*res_len);// derived
                        }
                else{
                        field_force = 0;        //particle is not in a component, then no force.
                }
        v[n][d]+=(field_force)/mass[n]*dt;//integrate again to update velocity with the acceleratio

        }
        else{
                field_force = 0;
                }

        }
```

Code added for graphing data Observing how current changes when different inputs changed was important to verify that the program was functioning properly.

Storing current data

```
//storing current values for graphing later
memmove(&IImeas[1],&IImeas[0],(MeasMax-1)*sizeof(double));
IImeas[0]=current_var;
```

Plotting current data with respect to time

```
DefineGraphN_R("Average Current",&IImeas[0],&Measlen,NULL);
```

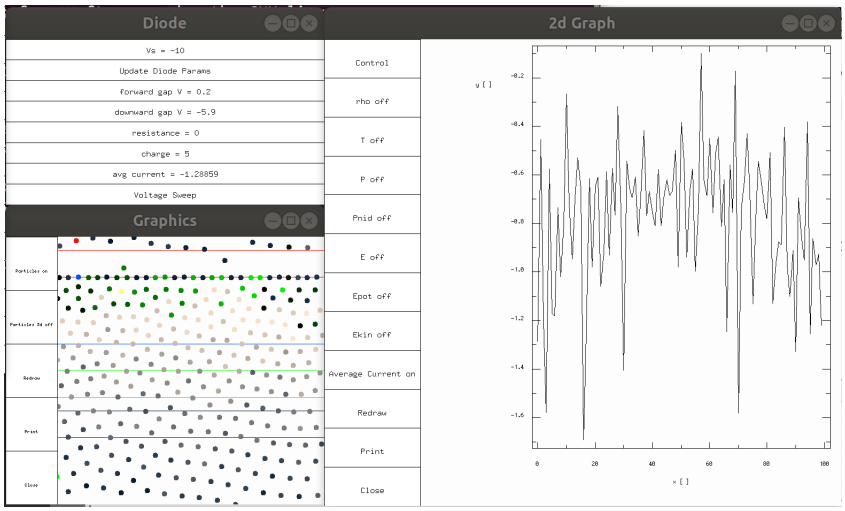

Figure 4: Diagram of doping process

Code added to GUI for running diode simulation

```
//menu code for diode midterm
StartMenu("Diode",0);
DefineDouble("Vs",&vs_v);
DefineFunction("Update Diode Params",setVoltageSource);
DefineDouble("forward gap V",&ud_v);
DefineDouble("downward gap V",&dd_v);
DefineDouble("resistance",&resistance);
DefineDouble("charge",&iq);
```

```
DefineDouble("avg current",&current_var);
StartMenu("Voltage Sweep",0);
DefineFunction("VI Curve",VI_Curve);
DefineDouble("V start",&v_start);
DefineDouble("V end",&v_end);
DefineDouble("v_incr",&v_incr);
DefineInt("v_sweep_delta",&v_sweep_delta);
EndMenu();
```

5 Procedure

6 Algorithms and Code

7 Results

1. Results for Forward-Biasing
   voltage sweep, -5V - 5V increments of 1v
2. Results for Reverse-Biasing voltage sweep -10V - 150V increments of 10v

Analyse the data your code is generating.
Do the results agree with your hypothesis?
How are the results different from your expectations?
What does that tell you about your project?
INSERT SCREENSHOTS OF CODE WORKING, GRAPHS, ETC

8 Conclusion

WHAT UNEXPECTED DIFFICULTIES DID YOU ENCOUNTER? HOW DID YOU ADDRESS THEM?

9 Individual Contribution

ADD BIT ABOUT HOW WORK WAS DIVIDED AMONG GROUP
Each group member will fill out the section next to their name.
Jacob Connelly:
David Jedynak:
Nick Zahler:

10 Code for Diode Simulation

```
/*
PHYS370 intro computational physics
March 20th, 2018
Miderm Project Diode Semiconductor
David Jedynak, Jacob Connelly, Nick Zaler

Purpose:
```

```
        Model a Diode using the molecular dynamics sim to observe how changing band g

Features:
        Complete -Tested
        - Diode make from 2 fields that allow particles to travel in one direction, a
        - Voltage source used to drive current through the diode
        - Real time Plotting of Current WRT time
        - A routine that sweeps voltage, measures current, and outputs the data to a
        - Inputs for forward voltage gap, Reverse votlage gap, volatge sweep(start,en
        - GUI window for inputs
        Complete - Untested
        - Resistor
        Incomplete
*/


#include <math.h>
#include <mygraph.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>

#define Nmax 1000
#define D 2
#define MeasMax 100



//variables added for midterm project diode
double current_var = 0;                         //global used for stoing the average
double field_force = 0;                         //for storing the force derived from
double g = 1,k = 1.1;                           //gravity and electric field constant.
double volt_len = 5,res_len = 5,diode_len_p = 5,diode_len_n = 5;//lengths of componen
double volt_pos = 45,res_pos = 15,diode_pos_p = 22.5,diode_pos_n = 27.5;//positions o
double resistance = 0;
double vs_v = -10,ud_v = 0.2,dd_v = -5.9;       //ideal voltages of diode band gaps
double v_f,ud_f,dd_f;                           //forces for the different fields
//voltage sweep
double v_start = 5.0,v_end = -5.0,v_incr = -1.0;//voltage sweep
int v_sweep_delta = 500;                        //delay for letting current settle ou

double  L=50;                                   // size of box

double mass[Nmax];                              // masses of the particles
double x[Nmax][D],v[Nmax][D];                   // State of the system
double q[Nmax];                                 // charges of the particles
double iq = 5;                                  //initial charge of particles

// parameters
double scalefac=100;
double x0[Nmax][D],v0[Nmax][D],dt=0.1,vv=0;     //initial conditions of system
double rho[MeasMax],Tset=0,Tmeas[MeasMax], ppnid[MeasMax],pp[MeasMax],Etot[MeasMax],E

int N=1000,Measlen=MeasMax,iterations=0;                //number of particles, length

// Global variables for Isotherm
int Thermalize=10000, MeasNo=1000;
```

```
// 3d visualization
double tet=0,phi=0,tetdot=0.05,phidot=0,shift=150;

// functions defined for midterm
double getCurrent(double v[N][D]){
        double sumV = 0;
        for(int n = 0;n<N;n++){
                sumV += v[n][1];
                }
        return iq*sumV/L;
}



/*The set density function changes the value L (box length) in order to achieve a give
void setdensity(){
  double fact=1/L;
  L=pow(N/rho[0],1./D);
  fact*=L;
  for (int n=0;n<N; n++)
    for (int d=0; d<D; d++)
      x[n][d]*=fact;
}

/*Calculate the temperature based off of the number of particles, kinetic energy, and
double density(){
  double r=0;
  r=N;
  for (int d=0; d<D; d++) r/=L;
  return r;
}
/*calculate the average Temperature by summing the kinetic energy, then dividing it by
double T(double v[N][D]){
  //double
  double t=0;
  for (int n=0; n<N; n++)
    for (int d=0; d<D; d++){
      t+=mass[n]*v[n][d]*v[n][d];
        }
  return t/N/D;
}
/*every time this is called the Temp gets set to whatever T is set to*/
void setTemp(){
  Tmeas[0]=T(v);
  double fact=sqrt(Tset/Tmeas[0]);
  for (int n=0;n<N; n++)
    for (int d=0; d<D; d++)
      v[n][d]*=fact;
}
/*Non ideal pressure*/
double Pnid(double x[N][D]){
  double p=0;
  for (int n=0; n<N; n++)
    for (int m=n+1; m<N; m++){
      double dr[D],dR=0;
      for (int d=0; d<D; d++){
        dr[d]=x[m][d]-(x[n][d]-L);
```

```
      double ddr;
      ddr=x[m][d]-(x[n][d]);
      if (fabs(ddr)<fabs(dr[d])) dr[d]=ddr;
      ddr=x[m][d]-(x[n][d]+L);
      if (fabs(ddr)<fabs(dr[d])) dr[d]=ddr;
      dR+=dr[d]*dr[d];
    }
    double dR6=dR*dR*dR;
    double dR12=dR6*dR6;
    double Fabs=12/dR12-6/dR6;
    p+=Fabs/D;
  }
  for (int d=0; d<D; d++) p/=L;

  return p;
}
/*Potential Energy, calculated using the forces and the distances between each partic
double Ep(double x[N][D],double v[N][D]){
  double EE=0;

  for (int n=0; n<N; n++)
    for (int m=n+1; m<N; m++){
      double dr[D];
      double dR=0;
      for (int d=0; d<D; d++){
        dr[d]=x[m][d]-(x[n][d]-L);
        double ddr;
        ddr=x[m][d]-(x[n][d]);
        if (fabs(ddr)<fabs(dr[d])) dr[d]=ddr;
        ddr=x[m][d]-(x[n][d]+L);
        if (fabs(ddr)<fabs(dr[d])) dr[d]=ddr;
        dR+=dr[d]*dr[d];
      }
      double dR6=dR*dR*dR;
      double dR12=dR6*dR6;
      EE += 1/dR12-1/dR6;
    }
  return 2*EE;
}
/*Calculates the force on each particle in each direction
-iterates though every particle calculates the distance between it and other particles
void F(double x[N][D], double v[N][D],double FF[N][D]){

  memset(&FF[0][0],0,N*D*sizeof(double)); //zeroize
  for (int n=0; n<N; n++) //iterate particles
    for (int m=n+1; m<N; m++){//
      double dr[D],dR=0;
      for (int d=0; d<D; d++){ //iterate dimensions
        dr[d]=x[m][d]-(x[n][d]-L);
        double ddr;
        ddr=x[m][d]-(x[n][d]);
        if (fabs(ddr)<fabs(dr[d])) dr[d]=ddr; // as the
        ddr=x[m][d]-(x[n][d]+L);
        if (fabs(ddr)<fabs(dr[d])) dr[d]=ddr; //as the particle gets far away force l
        dR+=dr[d]*dr[d];

      }
      double dR6=dR*dR*dR;
```

```
      double dR12=dR6*dR6;
      double Fabs=12/dR12/dR-6/dR6/dR + k*q[n]*q[m]/pow(dR,2) + g*mass[n]*mass[m]/pow
      for (int d=0;d<D; d++){
        FF[n][d]-=Fabs*dr[d];
        FF[m][d]+=Fabs*dr[d];
      }
    }
  return;
}
/*primary function, updates dynamics for particles using above functions and also con
-iterate through each velocity and integrate acceleration
-iterate position integrate velocity*/
void iterate(double x[N][D],double v[N][D],double dt){
  double ff[N][D];
  //itotal = 0;
  F(x,v,ff);
  if (iterations==0)
    for (int n=0;n<N;n++)
      for (int d=0;d<D;d++)
        v[n][d]+=0.5*ff[n][d]/mass[n]*dt;
  else
    for (int n=0;n<N;n++)
      for (int d=0;d<D;d++){
        //begin conditions for diode, resistor, and voltage source regions
        v[n][d]+=(ff[n][d])/mass[n]*dt;// integrate to get velocity

        if (d == 1){// if the dimension is in the Y dimension... vertical


                //check to see if the particle is in diode p, diode n, voltage, resist
                if ((x[n][d] < (diode_pos_p+diode_len_p/2)) && (x[n][d] > (diode_pos_p
                        field_force = ud_f - current_var;//simulate the forward diode
                        }
                else if((x[n][d] < (diode_pos_n+diode_len_n/2)) && (x[n][d] > (diode_p
                        field_force = dd_f + current_var;//simulate the reverse diode
                        }
                else if((x[n][d] < (volt_pos + volt_len/2)) && (x[n][d] > (volt_pos -
                        field_force = v_f;//simualate voltage source
                        }
                else if((x[n][d] > (res_pos - res_len/2)) && (x[n][d] < (res_pos + res
                        field_force = -v[n][d]*q[n]*q[n]*resistance/(res_len*res_len)
                        }
                else{
                        field_force = 0;        //particle is not in a component, ther
                }
        v[n][d]+=(field_force)/mass[n]*dt;//integrate again to update velocity with th

        }
        else{
                field_force = 0;
                }

        }

  current_var = getCurrent(v);          //update current value for measurements
  for (int n=0;n<N;n++)
    for (int d=0;d<D;d++){
      x[n][d]+=v[n][d]*dt;
```

```
      if (x[n][d]<0) x[n][d]+=L;
      else if (x[n][d]>=L) x[n][d]-=L;
    }
  setTemp();//added set temp to lower amout of mouse clicks for setting temperature.
  iterations++;
}

/*set initial velocities, masses*/
void setup(){
  int M=pow(N-1,1./D)+1;
  for (int n=0; n<N; n++){
    mass[n]=1;
    for (int d=0; d<D; d++){
      int nn=n;
      for (int dd=0; dd<d; dd++) nn/=M;
      x0[n][d]=(nn%M)*L/M;
      if (d==1){
        if (x0[n][0]<L/2)
          v0[n][d]=vv;
        else v0[n][d]=-vv;
      }
      else v0[n][d]=0;
    }
  }
}
void setVoltageSource(){
  //calculate forces for the given voltages
  v_f = iq*vs_v/volt_len;                //calculate the field from the set voltage
  ud_f = iq*ud_v/diode_len_p;            //calculate the field from the set upper diod
  dd_f = iq*dd_v/diode_len_n;            //calculate the field from the lower diode vo
  for (int n=0; n<N; n++){
    for (int d=0; d<D; d++){
    q[n] = iq;                           //set initial charge for each electron
    }}

}
/*sets initial positions and velocities for particles*/
void init(){
  for (int n=0; n<N; n++){
    for (int d=0; d<D; d++){
      x[n][d]=x0[n][d];
      v[n][d]=v0[n][d];
    }
    //set initial charge for each electron
    q[n] = iq;


}
  iterations=0;
}
/*saves the current state as the initial state. for future use. pretty nice*/
void GetState(){
  for (int n=0; n<N; n++)
    for (int d=0; d<N; d++){
      x0[n][d]=x[n][d];
      v0[n][d]=v[n][d];
    }
  iterations=0;
```

```
}
/*2D graph of particles*/
void draw(int xdim, int ydim){
  int size=xdim;
  if (ydim<size) size=ydim;
  scalefac=size/L;
//code added for midterm start
//add lines for notatiting the different accelaration fields
// color, x1,y1,x2,y2

//voltage source lines
  mydrawline(2,0,scalefac*(L-volt_pos+volt_len/2),size,scalefac*(L-volt_pos+volt_len/
  mydrawline(2,0,scalefac*(L-volt_pos-volt_len/2),size,scalefac*(L-volt_pos-volt_len/

//diode field 1 lines
  mydrawline(3,0,scalefac*(L-diode_pos_p + diode_len_p/2),size,scalefac*(L-diode_pos_p
  mydrawline(3,0,scalefac*(L-diode_pos_p - diode_len_p/2),size,scalefac*(L-diode_pos_p

//diode field 2 lines
 // mydrawline(4,0,100,size,100);
  mydrawline(4,0,scalefac*(L-diode_pos_n - diode_len_n/2),size,scalefac*(L-diode_pos_n

//resistor field lines
  mydrawline(6,0,scalefac*(L-res_pos+res_len/2),size,scalefac*(L-res_pos+res_len/2));
  mydrawline(6,0,scalefac*(L-res_pos-res_len/2),size,scalefac*(L-res_pos-res_len/2));

//code added for midterm end

  mydrawline(1,0,size,size,size);
  mydrawline(1,size,0,size,size);
  for (int n=0; n<N; n++){
    int xx=x[n][0]*scalefac;
    int yy=x[n][1]*scalefac;
    myfilledcircle(n+1,xx,size-yy,0.5*scalefac);
  }
}
/**/
int compare(const void *x1,const void *x2){
  if (((double *) x1)[2]< ((double *)x2)[2]) return 1;
  else return -1;
}
/*3D graph of particles*/
void draw3d(int xdim, int ydim){
  int size=xdim;
  typedef struct part{
    double x[D]; // position
    int c; //color
  } part;
  part p[N];

  tet+=tetdot;                                                    //increment th
  phi+=phidot;                                                    //increment th

  if (D!=3){
    printf("3d visualization requires D=3, you have D=%i!",D);
    return;
  }
  if (ydim<size) size=ydim;
```

```
  scalefac=0.7*size/L*shift;

  // center cube
  for (int n=0; n<N; n++)
    for (int d=0; d<D; d++){
      p[n].x[d]=x[n][d]-L/2;
      p[n].c=n+1;
    }
  // rotate around y
  double c=cos(tet);
  double s=sin(tet);
  for (int n=0; n<N; n++){
    double x=c*p[n].x[0]+s*p[n].x[2];
    p[n].x[2]=-s*p[n].x[0]+c*p[n].x[2];
    p[n].x[0]=x;
  }
  // rotate around x
  c=cos(phi);
  s=sin(phi);
  for (int n=0; n<N; n++){
    double y=c*p[n].x[1]+s*p[n].x[2];
    p[n].x[2]=-s*p[n].x[1]+c*p[n].x[2];
    p[n].x[1]=y;
  }
  // shift box away from origin
  for (int n=0; n<N; n++){
    p[n].x[2]+=shift;
  }
  qsort(&p[0].x[0],N,sizeof(part),&compare);

  for (int n=0; n<N; n++){                                    //draw the circles wi
    int xx=p[n].x[0]/p[n].x[2]*scalefac;
    int yy=p[n].x[1]/p[n].x[2]*scalefac;
    myfilledcircle(0      ,xdim/2+xx,ydim/2-yy,0.5/p[n].x[2]*scalefac+2);
    myfilledcircle(p[n].c,xdim/2+xx,ydim/2-yy,0.5/p[n].x[2]*scalefac);
  }
}
/*store data from simulation for various dynamics*/
void Measure(){
  memmove(&rho[1],&rho[0],(MeasMax-1)*sizeof(double));
  rho[0]=density();
  memmove(&Tmeas[1],&Tmeas[0],(MeasMax-1)*sizeof(double));
  Tmeas[0]=T(v);
  //storing current values for graphing later
  memmove(&IImeas[1],&IImeas[0],(MeasMax-1)*sizeof(double));
  IImeas[0]=current_var;
  memmove(&ppnid[1],&ppnid[0],(MeasMax-1)*sizeof(double));
  ppnid[0]=Pnid(x);
  memmove(&pp[1],&pp[0],(MeasMax-1)*sizeof(double));
  pp[0]=rho[0]*Tmeas[0]+ppnid[0];
  memmove(&Ekin[1],&Ekin[0],(MeasMax-1)*sizeof(double));
  Ekin[0]=N*D*Tmeas[0];
  memmove(&Epot[1],&Epot[0],(MeasMax-1)*sizeof(double));
  Epot[0]=Ep(x,v);
  memmove(&Etot[1],&Etot[0],(MeasMax-1)*sizeof(double));
  Etot[0]=Epot[0]+Ekin[0];

}
```

```
/*isotherm file management / writing*/
void Isotherm(){
  FILE *res;
  char IsoName[100];

  sprintf(IsoName,"Iso%f_%i.dat",Tset,N);
  res=fopen(IsoName,"w");

  for (rho[0]=density(); rho[0]>0.01;rho[0]/=1.1){
    setdensity();
    // thermalize
    for (int i=0; i<Thermalize; i++){
      setTemp();
      iterate(x,v,dt);
    }
    // measure values
    double PP=0, TT=0;
    for (int i=0; i<MeasNo; i++){
      iterate(x,v,dt);
      double Tmeas=T(v);
      TT+=Tmeas;
      PP+=rho[0]*Tmeas+Pnid(x);
    }
    TT/=MeasNo;
    PP/=MeasNo;
    fprintf(res,"%e %e %e\n",rho[0],PP,TT);
    Events(1);
    DrawGraphs();
  }
  fclose(res);
}
/*isotherm routine*/
void Isotherms(){
  double LStart=L;
  for (Tset=0.05; Tset<2; Tset+=0.05){
    L=LStart;
    init();
    Isotherm();
  }
}

/*This routine sweeps the voltage from start to end with an increment, outputs a file
void VI_Curve(){
  FILE *res;
  char IsoName[100];
  double sumI = 0;

  sprintf(IsoName,"VI_Curve_Q_%f_N_%i_ud_v_%f_dd_v_%f.dat",iq,N,ud_v,dd_v);//put info
  res=fopen(IsoName,"w");
  //sweep the voltage
  if(v_incr < 0 ){
          for (double ivolt = v_start;ivolt>v_end;ivolt = ivolt + v_incr){
                //set the voltage
                vs_v = ivolt;
                setVoltageSource();
                //loop to waste some time to get the current to settle
                for(int t = 0;t<v_sweep_delta;t++)
                        {
```

```
                        iterate(x,v,dt);
                        Events(1);
                        DrawGraphs();
                        sumI += current_var;
                        }
                //save the current and voltage data
                fprintf(res,"%e %f\n",sumI/v_sweep_delta,vs_v);
            }
    }
      else{
            for (double ivolt = v_start;ivolt<v_end;ivolt = ivolt + v_incr){
                //set the voltage
                vs_v = ivolt;
                setVoltageSource();
                //loop to waste some time to get the current to settle
                for(int t = 0;t<v_sweep_delta;t++)
                        {
                        iterate(x,v,dt);
                        Events(1);
                        DrawGraphs();
                        sumI += current_var;
                        }
                //save the current and voltage data
                fprintf(res,"%e %f\n",sumI/v_sweep_delta,vs_v);
            }
    }

  fclose(res);
}



/**/
int main(){
  struct timespec ts={0,1000000};
  int cont=0;
  int sstep=0;
  int repeat=10;
  int done=0;
  char name[50],mname[N][50];
  setup();
  init();
  Measure();

  DefineGraphN_R("rho",&rho[0],&Measlen,NULL);
  DefineGraphN_R("T",&Tmeas[0],&Measlen,NULL);
  DefineGraphN_R("P",&pp[0],&Measlen,NULL);
  DefineGraphN_R("Pnid",&ppnid[0],&Measlen,NULL);
  DefineGraphN_R("E",&Etot[0],&Measlen,NULL);
  DefineGraphN_R("Epot",&Epot[0],&Measlen,NULL);
  DefineGraphN_R("Ekin",&Ekin[0],&Measlen,NULL);
  DefineGraphN_R("Average Current",&IImeas[0],&Measlen,NULL);

  AddFreedraw("Particles",&draw);
  AddFreedraw("Particles 3d",&draw3d);
  StartMenu("Newton",1);
  DefineMod("No part",&N,Nmax);
  DefineDouble("L",&L);
```

```
DefineDouble("dt",&dt);
StartMenu("measure",0);
DefineDouble("Average Current",&IImeas[0]);
DefineDouble("rho",&rho[0]);
DefineFunction("setrho",setdensity);
DefineDouble("T",&Tmeas[0]);
DefineDouble("T set",&Tset);
DefineFunction("set T",setTemp);
DefineDouble("P",&pp[0]);
DefineDouble("Pnid",&ppnid[0]);
EndMenu();
StartMenu("Isotherm",0);
DefineInt("Thermalize",&Thermalize);
DefineInt("MeasNo",&MeasNo);
DefineFunction("Measure Isotherm",Isotherm);
DefineFunction("Measure multiple Isotherms",Isotherms);
EndMenu();
StartMenu("init",0);
for (int n=0; n<N; n++){
}
if (N<15)
  for (int n=0; n<N; n++){
    sprintf(mname[n],"Particle %i",n);
    StartMenu(mname[n],0);
    DefineDouble("m",&mass[n]);
    for (int d=0; d<D; d++){
      sprintf(name,"x[%i]",d);
      DefineDouble(name,&x0[n][d]);
    }
    for (int d=0; d<D; d++){
      sprintf(name,"v[%i]",d);
      DefineDouble(name,&v0[n][d]);
    }
    EndMenu();
  }
DefineDouble("vv",&vv);
DefineFunction("setup",&setup);
DefineFunction("Get State",&GetState);
DefineFunction("init",&init);
EndMenu();
//menu code for diode midterm
StartMenu("Diode",0);
DefineDouble("Vs",&vs_v);
DefineFunction("Update Diode Params",setVoltageSource);
DefineDouble("forward gap V",&ud_v);
DefineDouble("downward gap V",&dd_v);
DefineDouble("resistance",&resistance);
DefineDouble("charge",&iq);
DefineDouble("avg current",&current_var);
StartMenu("Voltage Sweep",0);
DefineFunction("VI Curve",VI_Curve);
DefineDouble("V start",&v_start);
DefineDouble("V end",&v_end);
DefineDouble("v_incr",&v_incr);
DefineInt("v_sweep_delta",&v_sweep_delta);
EndMenu();
EndMenu();
DefineGraph(curve2d_,"Measurements");
```

```
  DefineDouble("phi",&phi);
  DefineDouble("phidot",&phidot);
  DefineDouble("tet",&tet);
  DefineDouble("tetdot",&tetdot);
  DefineDouble("shift",&shift);
  DefineGraph(freedraw_,"graph");
  DefineInt("repeat",&repeat);
  DefineBool("sstep",&sstep);
  DefineLong("NS_slow",&ts.tv_nsec);
  DefineBool("cont",&cont);
  DefineBool("done",&done);
  EndMenu();
  while (!done){
    Events(1);
    DrawGraphs();
    if (cont||sstep){
      sstep=0;
      for (int i=0; i<repeat; i++) iterate(x,v,dt);
      Measure();
    }
    else        nanosleep(&ts,NULL);
  }
}
```

11 References

https://www.ndsu.edu/pubweb/ carswagn/LectureNotes/370/index.html