Carleton University

Modelling of Integrated Devices

ELEC 4700

Assignment 1

Author: Maharshi Gurjar
Student number: 101094330
Date submitted: February 8, 2021

Contents

List of Figures

List of Source Codes

1 Electron Modelling

Thermal Energy In this simulation the thermal energy is calculated using Maxwell's principle of equipartition of energy, with the consideration that the simulation is running in 2D. This equation is given as below,

$$\overline{KE} = \frac{1}{2}kT = 2\left(\frac{1}{2}m\overline{v^2}\right) \implies \overline{v^2} = \frac{2kT}{m} \tag{1}$$

Thermal Velocity The thermal velocity (V_{th}) as shown given in the equation above was calculated using the simulation parameter of T = 300K, equating to,

$$\overline{V_{th}} = \frac{2 \times (1.38064852 \times 10^{-23}) \times 300}{0.26 \times (9.10938356 \times 10^{-31})} = 1.8702 \times 10^{05} \tag{2}$$

or simply, 187 km/s.

Mean Free Path (L) Given a mean time between collisions ($\tau_{mn} = 0.2$ps), the mean time between collisions is given as,

$$L = V_{th} \times 0.2 \times 10^{-12} = 3.7404 \times 10^{-08} \tag{3}$$

1.1 Control Variables

Source Code 1 below shows the initialization of the control variables used in part 1. **Show_movie** can be set to either 1 (to show the movement of electrons) or 0 (to show only the final simulation results). Additionally the top and bottom boundary's of the region can be set to specular or diffusive by changing lines 23 and 24 (1 for specular and 0 for diffusive).

```
1  %{
2  % Author: Maharshi Gurjar
3  % ELEC 4700 - Modeling of Integrated Devices
4  % Assignment 1
5  %}
6  clc; close all; clear;
7  set(0, 'DefaultFigureWindowStyle', 'docked')
8  %Define simulation environment and constants
9  M0 = 9.10938356e-31; %Rest mass of electron
10 Mass_n = 0.26*M0; %Effective mass of electron
11 T = 300; % Simulation environment temperature (K)
12 k = 1.38064852e-23; % Boltzmans constant
13 V_thermal = sqrt(2*k*T/Mass_n) %Thermal Velocity
14 Height = 100e-9; % The height of the simulation environment
15 Length = 200e-9; % The length of the simulation environment
16 nElectrons = 2e3; % Total number of electrons to simulate
17 nPlotted_Electrons = 10; %Total number of electrons displayed
18 Time_Step = Height/V_thermal/100; % Time step of simulation
19 Iterations = 1000; % Number of iterations to simulate
20 Show_Movie = 0; %Display steps control
21 % The mean free path is determined by multiplying the thermal velocity
22 ... by the mean time between collisions:
23 MFP = V_thermal * 0.2e-12 %Mean free path
24 %Temperature will be recorded in the array below
25 Temperature = zeros(Iterations,1);
26 %Setting the top/bottom of the boxes specularity
27 Top_Specular = 1;
28 Bottom_Specular = 1;
```

Source Code 1: Control variables for the simulation

1.2 Generating electrons

The states of all electrons are stored in a (N,4) matrix, where N is a unique electron and columns [1 4] represent the electrons positions and velocities [x y Vx Vy]. By preallocating the matrix size, the initial generation of the matrix is quicker, where electrons x and y positions are calculated as the shown in Source Code 2 below.

```
29 %The state of the electron  (position and velocity) is stored in a array
30 ... where each index refers to [x-position y-position v-in-x v-in-y]
31 Electron_State = zeros(nElectrons,4);
32 %Generate a random initial population position and velocity
33 for i = 1:nElectrons
34    Electron_State(i,:) = [Length*rand() Height*rand() V_thermal*cos(rand()*2*pi)
      ↪  V_thermal*sin(rand()*2*pi)];
35 end
```

Source Code 2: Generating the electrons initial spacial characteristics

1.3 Simulating the movement of electrons

The movement of each electron was done using a single for loop to reduce the resource usage (with single for loop it becomes O(N) in Big-O notation[1]) with the use of specialized Matlab matrix indexing/equations. The first important aspect done was the movement of the electron as shown on line (41) in Source Code 3 below.

```
36 %We will now move (iterate) over time, updating the positions and direction
37 ...while plotting the state
38 for i = 1:Iterations
39     %The line below updates the x,y position by moving it to a new position
40     ... using its current position + the velocity*(time step)
41     Electron_State(:,1:2) = Electron_State(:,1:2) + Time_Step.*Electron_State(:,3:4);
42
43     %Checking boundary conditions using Matlab matrix equations
44
45     %Check if and move all electrons at X=200nm Bound:
46     Electron_State((Electron_State(:,1)>Length),1) = Electron_State((Electron_State(:,1)>Length),1)
       ↪  - Length;
47
48     %Check if and move all electrons at X=0nm Bound:
49     Electron_State((Electron_State(:,1)<0),1) =Electron_State((Electron_State(:,1)<0),1) + Length;
50
51     %Check (if) and move all electrons at Y Bounds and if specular or diffusive:
52     if (Top_Specular == 1)
53        Electron_State((Electron_State(:,2)>Height),4) =
          ↪  -1*Electron_State((Electron_State(:,2)>Height),4) ;
54        Electron_State((Electron_State(:,2)>Height),2) = 2*Height -
          ↪  Electron_State((Electron_State(:,2)>Height),2);
55     else
56        Electron_State((Electron_State(:,2)>Height),4) = -random(Velocity_PDF);
57        Electron_State((Electron_State(:,2)>Height),3) = random(Velocity_PDF);
58     end
59     if (Bottom_Specular == 1)
60         Electron_State((Electron_State(:,2)<0),4) = -1*Electron_State((Electron_State(:,2)<0),4) ;
61         Electron_State((Electron_State(:,2)<0),2) = -Electron_State((Electron_State(:,2)<0),2);
62     else
63        Electron_State((Electron_State(:,2)>Height),4) = random(Velocity_PDF);
64        Electron_State((Electron_State(:,2)>Height),3) = random(Velocity_PDF);
65     end
```

[1] This applies for only the electron movement

```
66      % Store the electron trajectories in two individual matrix
67      for j = 1: nPlotted_Electrons
68         Trajectories_x(i,j) = Electron_State(j,1);
69         Trajectories_y(i,j) = Electron_State(j,2);
70      end
71      %To calculate the thermal energy, Maxwell's principle of equipartion
72      ... is used,  where the final equation then becomes;
73      Temperature(i) = ( sum (Electron_State(:,3).^2) + sum(Electron_State(:,4).^2)) * Mass_n / k / 2
        ↪  / nElectrons;
74
75      %Shows the pathing of the electron, as well as the updating trajectory
76      if Show_Movie && mod(i,50)
77         figure(1)
78         hold off;
79         plot(Electron_State(1:nPlotted_Electrons,1)./1e-9,Electron_State(1:nPlotted_Electrons,2)./1e
           ↪  -9,'o');
80         grid on;
81         axis([0 Length/1e-9 0 Height/1e-9]);
82         xlabel('x (nm)');
83         ylabel('y (nm)');
84         title(sprintf("Plotting (%d/%d) electron at constant
           ↪  velocity",nPlotted_Electrons,nElectrons));
85         hold on;
86      end
87   end
```

Source Code 3: Main iterative loop for simulating electron movement

Boundary checks The code will then check if the electrons current state is outside X bounds [0 100nm], and if show it will move all of the electrons in the matrix to their proper location. The same is applied for the Y bounds, with the inclusion of a diffusive or specular characteristic, wherein if the boundary is specular a reflection occurs and if diffusive a random velocity is assigned to the particle (Lines 52-65).

Trajectories, temperature and movie Individual electron x and y positions are then stored in thier respective matrices as shown on lines (67-70). The temperature calculation on line (73) is done to calculate the total temperature of the simulated region. Finally, if the **Show_movie** condition is enabled, every 50 frames a figure will update showing the electrons new positions.

1.4 Plotting simulation data

The final section of the Part 1 code handles the plotting of the simulation results, as shown in Source Code 4 below.

```
88   figure("name","Trajectory, temperature and speed results results")
89   subplot(2,1,1)
90   hold on;
91   plot(Trajectories_x(:,1:nPlotted_Electrons)./1e-9, Trajectories_y(:,1:nPlotted_Electrons)./1e-9,'.'
     );
92   hold off;
93   axis([0 Length/1e-9 0 Height/1e-9]);
94   xlabel('x (nm)');
95   ylabel('y (nm)');
96   grid on;
97   title(sprintf("Trajectories of (%d/%d) electron(s) at constant
     ↪  velocity",nPlotted_Electrons,nElectrons));
98
99   subplot(2,1,2)
100  plot(Time_Step*(0:Iterations-1), Temperature);
```

```
grid on;
xlim([0 Time_Step*Iterations])
title(sprintf("Temperature of the region, Average Temperature: %.2f",mean(Temperature)))
xlabel('Time (s)');
ylabel('Temperature (K)');
```

Source Code 4: Plotting the finalized simulation results

1.5 Part 1 Simulation results

Figure 1 below shows the results of the simulation, where the first sub figure shows the trajectories of the electrons (shows a preset (x number of electrons)/(total number of electrons)) and the second sub figure shows the temperature of the region.

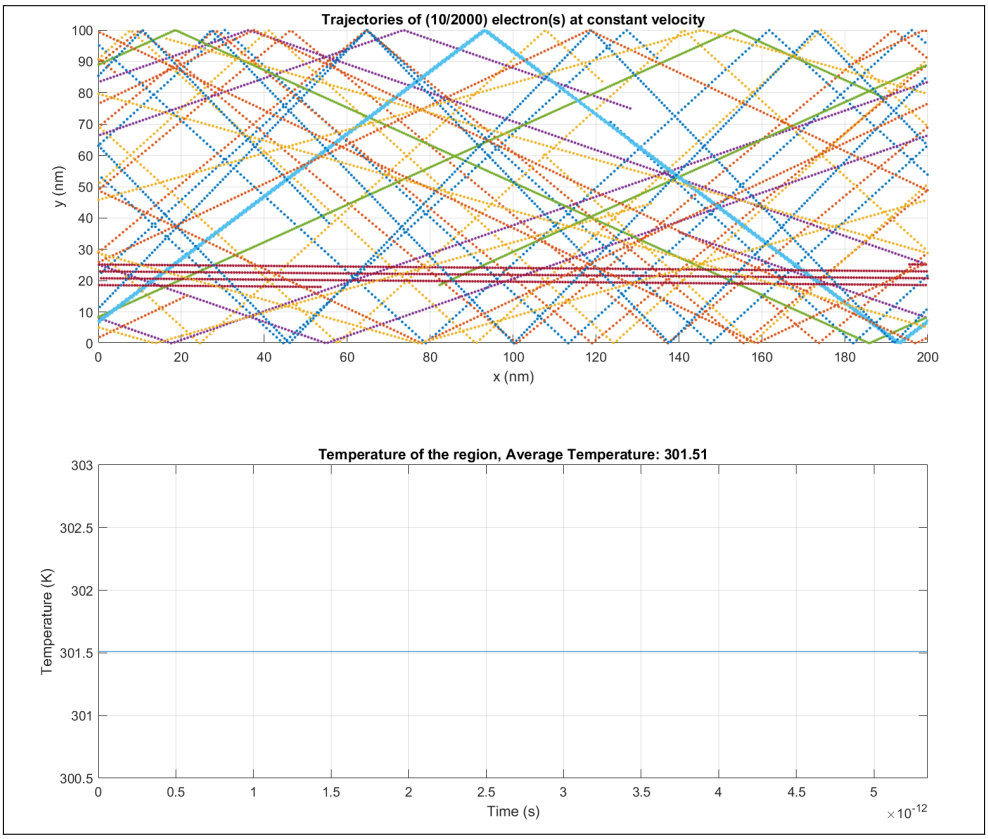


Figure 1: Simulation results from Part 1

The results of the simulation show the expected relationship. If the electron makes contact with the top or bottom boundary of the region it gets reflected, and if the electron makes contact with the left or right boundary it moves to the other side. An observation can be made towards the temperature of the environment, without the random movement of electrons (all electrons move at a constant velocity at all times) the temperature remains at a constant value.

2 Collisions with Mean Free Path (MFP)

To avoid redundancy, **Part 2** and **Part 3** will only include any changes/additions to the code as given in **Part 1**, as the code in each part builds up on the previous.

2.1 Including scattering effect

The second portion of this assignment involves introducing the scattering effect on the movement of the electrons. Scattering occurs the probability given by,

$$\mathrm{P}_{\mathrm{Scattering}} = 1 - e^{\frac{dt}{\tau_{mn}}} = 0.0264 \tag{4}$$

Additionally, the velocity of the electrons is generated with a Gaussian distribution with a standard deviation of $\sqrt{kT/m}$. This is implemented in matlab using the 'makedis' function as shown in Source Code 5 below.

```
32 %Create a scattering probability
33 P_Scatterieng = 1 - exp(-Time_Step/0.2e-12);
34 %Create a distribution using the matlab makedist function
35 Velocity_PDF = makedist('Normal', 'mu', 0, 'sigma', sqrt(k*T/Mass_n));
36 %Generate a random inital population postion and velocity
37 for i = 1:nElectrons
38    Electron_State(i,:) = [Length*rand() Height*rand() random(Velocity_PDF) random(Velocity_PDF)];
39 end
```

Source Code 5: Scattering probability and Gaussian distribution

2.2 Scattering during simulation

The scattering effect during simulation is implemented by simply adding two lines to the main iterative loop, as shown Source Code 6 below,

```
70 %Add scattering
71 j = rand(nElectrons,1) < P_Scatterieng;
72 Electron_State(j,3:4) = random(Velocity_PDF,[sum(j),2]);
```

Source Code 6: Scattering the movement of electrons during the simulation

2.3 Plotting speed histogram

Finally, a histogram in the simulation results figure was added using the matlab 'histogram' function as shown in Source Code 7 below.

```
116 subplot(3,1,3)
117 Velocity = sqrt(Electron_State(:,3).^2 + Electron_State(:,4).^2);
118 histogram(Velocity);
119 title(sprintf("Electron Velocity, Average Velocity: %.2d",mean(Velocity)));
120 xlabel("Speed (m/s)");
121 ylabel("Number of particles");
122 grid on;
```

Source Code 7: Adding histogram to simulation results

2.4 Part 2 Simulation results

Figure 2 below shows the results of the simulation, with the inclusion of the 'speed histogram'. Comparing these results to those in **Part 1** (Figure 1) shows the effects of scattering. Where the temperature previously was constant, due to scattering there is now a fluctuation, which creates a higher total average temperature.

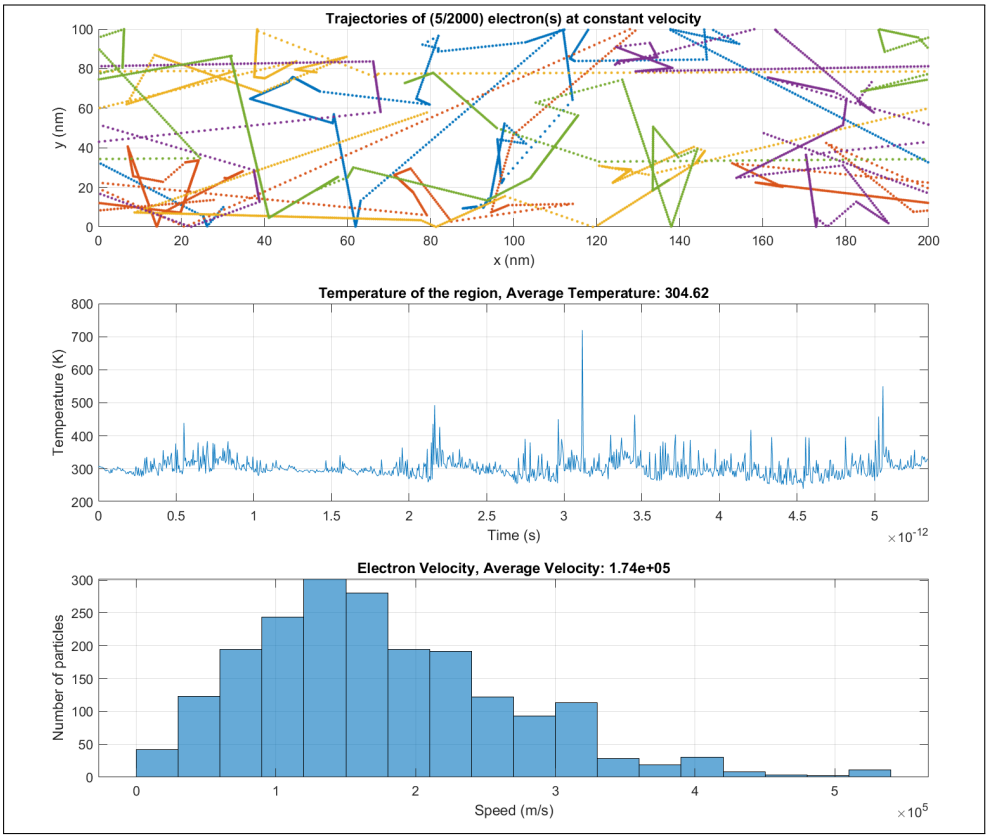


Figure 2: Simulation results from Part 2

Diffusive boundary (y=100nm) Additionally, this simulation was ran with the top boundary being diffusive to produce some interesting results. In sub figure 2 (temperature graph) there are instances where the temperature spikes by a large amount, this is due to a number of electrons bouncing off of the top layer at very high speeds. Conversely between 1 - 1.5 pS there exist a period where the temperature is relatively stable, caused by the same effect wherein many electrons do not make contact with the diffusive boundary.

3 Enhancements

The goal final portion of the assignment was to add in an inner rectangle 'bottle neck' boundary, as shown in Figure 3. Electrons must not exist inside of these boxes, which means the conditions of the initial generation must be changed, and a boundary behaviour must be added to the box regions.

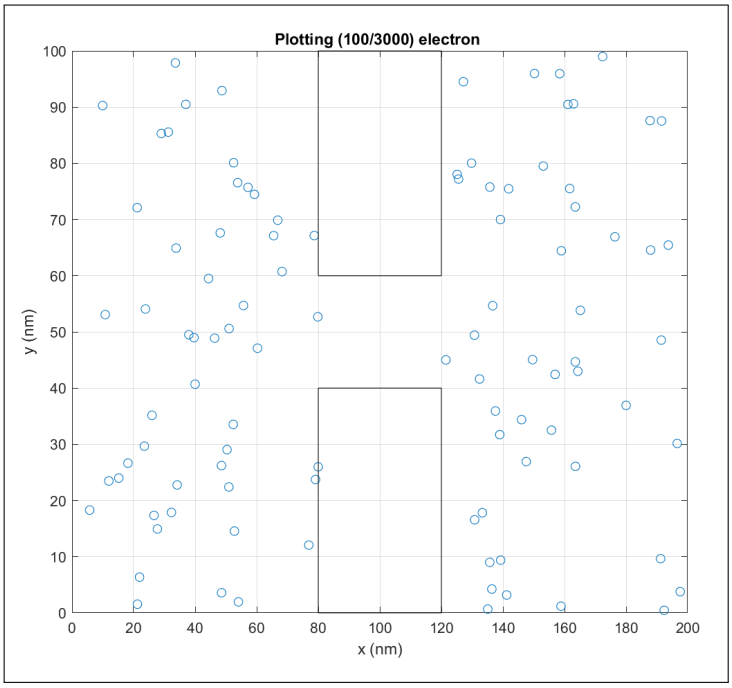


Figure 3: Rectangular bottle neck

3.1 Create boxes and initialize electrons

As mentioned above, when generating the initial spacial characteristics of the electrons the condition of existing outside of the boxes must be added. Once again using matrix indexing/equations the initial for loop is modified to do so, as shown in 8 below.

```
36 %Create Box-positions [x1, x2, y1,y2]
37 Box_pos = 1e-9.*[80 120 0 40; 80 120 60 100];
38 %Create the state of the box (specular or diffusive)
39 %Generate a random initial population position and velocity
40 for i = 1:nElectrons
41    Electron_State(i,:) = [Length*rand() Height*rand() random(Velocity_PDF) random(Velocity_PDF)];
42    Electron_State( (Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 & ...
43        (Electron_State(:,2)<40e-9)) ,1) = Length*rand();
44    Electron_State( (Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 & ...
45        (Electron_State(:,2)>60e-9)) ,1) = Length*rand();
46 end
```

Source Code 8: Checking box boundary conditions during electron initialization

The condition which checks the electron position reads as follows,

```
if (80nm<x-position<120m AND (y-position<40nm))
   move all electrons to random x-position
end
if (80nm<x-position<120m AND (y-position>60nm))
   move all electrons to random x-posistion
end
```

3.2 Box boundary conditions

This is the core section of the code which determines the behaviour of the electron as it makes contact with the boxes. This section of the code will be explained by breaking it up into two portions, one that deals with the boxes if they were specular, and the other if they were diffusive. The behaviour of the code is fundamentally the same, the response it provides to the simulation results in a few different results. The box characteristics are firstly defined, with 1 being specular and 0 being diffusive as shown below in Source Code 9.

```
36 %Set the box characteristics (0 = diffusive)
37 Top_Box_Specular = 1;
38 Bottom_Box_Specular = 1;
```

Source Code 9: Defining box boundary characteristic

3.2.1 Specular box boundaries

The first case considered is the specular case, where if an electron collides with the box boundary it simply bounces back (velocity in is equal to velocity out). Source Code 10 below shows how this was implemented[2].

```
97  %BOX SPECULAR CASE :
98  %Check if bottom Box is specular and if so then bounce
99  Electron_State( (Bottom_Box_Specular==1 & Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 &
    ↪  ((Electron_State(:,2)<40e-9 & Electron_Prev_State(:,2)<40e-9))),3)...
100     =-Electron_State( (Bottom_Box_Specular==1 & Electron_State(:,1)>80e-9 &
        ↪  Electron_State(:,1)<120e-9 & ((Electron_State(:,2)<40e-9 & ...
101     Electron_Prev_State(:,2)<40e-9))),3);
102 %Check if top Box is specular and if so then bounce
103 Electron_State( (Top_Box_Specular==1 & Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 &
    ↪  ((Electron_State(:,2)>60e-9 & Electron_Prev_State(:,2)>60e-9))),3)...
104     =-Electron_State( (Top_Box_Specular==1 & Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9
        ↪  & ((Electron_State(:,2)>60e-9 & Electron_Prev_State(:,2)>60e-9))),3);
105
106 %Top Horizontal
107 Electron_State( (Top_Box_Specular==1 & Electron_State(:,1)<120e-9 & Electron_State(:,1)>80e-9) &
    ↪  (Electron_State(:,2)>60e-9) &...
108     (Electron_Prev_State(:,1)>80e-9 & Electron_Prev_State(:,1)<120e-9),4) = -Electron_State(
        ↪  (Top_Box_Specular==1 & Electron_State(:,1)<120e-9...
109     & Electron_State(:,1)>80e-9) & (Electron_State(:,2)>60e-9) &(Electron_Prev_State(:,1)>80e-9 &
        ↪  Electron_Prev_State(:,1)<120e-9),4);
110 %Bottom Hoizontal
111 Electron_State( (Bottom_Box_Specular==1 & Electron_State(:,1)<120e-9 & Electron_State(:,1)>80e-9) &
    ↪  (Electron_State(:,2)<40e-9) &...
112     (Electron_Prev_State(:,1)>80e-9 & Electron_Prev_State(:,1)<120e-9),4) = -Electron_State(
        ↪  (Bottom_Box_Specular==1 & Electron_State(:,1)<120e-9...
113         & Electron_State(:,1)>80e-9) & (Electron_State(:,2)<40e-9) &(Electron_Prev_State(:,1)>80e-9
            ↪  & Electron_Prev_State(:,1)<120e-9),4);
114     %BOX SPECULAR CASE END :
```

Source Code 10: Box boundary - Specular

The logic used to define the box boundaries for specular is the same as for diffusive, with the only difference being the assignment of a new velocity. For specular as shown in the code above, the velocity is simply the negative component. The velocity assignment for diffusive will be discussed in the following section.

[2]Note: The code from this point out is much easier to read in Matlab itself, as the page margins can only be so wide

The condition which checks the electron position reads as shown below[3]. This condition is applied twice, once for each box and is **only** active when the the specific box is set to specular.

```
if ((Box==Specular) AND (current position is inside box)
        AND (previous y-position<40nm))
   Bounce electron (change vx)
end
if ((Box==Specular AND (current position is inside box)
        AND (40nm<y-position<60nm)
    Bounce electron (change vy)
end
```

3.2.2 Diffusive box boundaries

The second case that is considered is the diffusive case, where if an electron collides with the box boundary is bounces at a random velocity. Source Code 11 below shows how this was implemented.

```
116 %BOX DIFFUSIVE CASE:
117 %Check if bottom Box is specular and if so then bounce
118 Electron_State( (Bottom_Box_Specular==0 & Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 &
    ↪  ((Electron_State(:,2)<40e-9 & Electron_Prev_State(:,2)<40e-9))),3)...
119     =random(Velocity_PDF);
120 Electron_State( (Bottom_Box_Specular==0 & Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 &
    ↪  ((Electron_State(:,2)<40e-9 & Electron_Prev_State(:,2)<40e-9))),4)...
121     =random(Velocity_PDF);
122
123 %Check if Top Box is specular and if so then bounce
124 Electron_State( (Top_Box_Specular==0 & Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 &
    ↪  ((Electron_State(:,2)>60e-9 & Electron_Prev_State(:,2)>60e-9))),3)...
125     =random(Velocity_PDF);
126 Electron_State( (Top_Box_Specular==0 & Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 &
    ↪  ((Electron_State(:,2)>60e-9 & Electron_Prev_State(:,2)>60e-9))),4)...
127     =random(Velocity_PDF);
128
129 %Bottom Horizontal
130 Electron_State( (Bottom_Box_Specular==0 & Electron_State(:,1)<120e-9 & Electron_State(:,1)>80e-9) &
    ↪  (Electron_State(:,2)<40e-9) &...
131     (Electron_Prev_State(:,1)>80e-9 & Electron_Prev_State(:,1)<120e-9),3) = random(Velocity_PDF);
132 Electron_State( (Bottom_Box_Specular==0 & Electron_State(:,1)<120e-9 & Electron_State(:,4)<0 &
    ↪  Electron_State(:,1)>80e-9) & (Electron_State(:,2)<40e-9) &...
133     (Electron_Prev_State(:,1)>80e-9 & Electron_Prev_State(:,1)<120e-9),4) = random(Velocity_PDF);
134 %Top Horizontal
135 Electron_State( (Top_Box_Specular==0  & Electron_State(:,1)<120e-9 & Electron_State(:,1)>80e-9) &
    ↪  (Electron_State(:,2)>60e-9) &...
136     (Electron_Prev_State(:,1)>80e-9 & Electron_Prev_State(:,1)<120e-9),3) = random(Velocity_PDF);
137 Electron_State( (Top_Box_Specular==0 & Electron_State(:,4)>0 & Electron_State(:,1)<120e-9 &
    ↪  Electron_State(:,1)>80e-9) & (Electron_State(:,2)>60e-9) &...
138     (Electron_Prev_State(:,1)>80e-9 & Electron_Prev_State(:,1)<120e-9),4) = -random(Velocity_PDF);
```

Source Code 11: Box boundary - Diffusive

The conditions for velocity change are the same as in the specular case, but as previously mentioned the velocity is adjusted differently. Both the Vx and Vy of the electron are now assigned a random velocity for the vertical box boundaries. A special consideration must be given to the horizontal box boundaries, as the new random velocity must not make the electron travel into the box. This special case is easily resolved by simply forcing the new random velocity for the top box horizontal boundary to be negative, ensuring the electron cannot bounce into the box.

[3] Condition as shown is for the bottom box

3.2.3 Electron Shifting

The final consideration that must be had is the electron shifting with respect to the box boundaries. As with **Part 1**, which did the same process for electrons outside of the box boundary the same process must be done in this part.

```
144 %Electron shifting
145 %Region 1
146 Electron_State( (Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 &
    ↪  ((Electron_State(:,2)<40e-9 & Electron_Prev_State(:,2)<=40e-9)) &...
147     Electron_Prev_State(:,1)<=80e-9),1) = 2*80e-9 -Electron_State( (Electron_State(:,1)>80e-9 &
        ↪  Electron_State(:,1)<120e-9 ...
148     & ((Electron_State(:,2)<=40e-9 & Electron_Prev_State(:,2)<40e-9)) &
        ↪  Electron_Prev_State(:,1)<80e-9),1);
149 %Region 2
150 Electron_State( (Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 &
    ↪  ((Electron_State(:,2)<40e-9 & Electron_Prev_State(:,2)<=40e-9)) &...
151     Electron_Prev_State(:,1)>120e-9),1) = 2*120e-9 - Electron_State( (Electron_State(:,1)>80e-9 &
        ↪  Electron_State(:,1)<120e-9 ...
152     & ((Electron_State(:,2)<=40e-9 & Electron_Prev_State(:,2)<40e-9)) &
        ↪  Electron_Prev_State(:,1)>120e-9),1);
153 %Region 3
154 Electron_State( ( (Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9) &
    ↪  ((Electron_State(:,2)>60e-9 & Electron_Prev_State(:,2)>=60e-9)) &...
155     Electron_Prev_State(:,1)<80e-9),1) = 2*80e-9 - Electron_State( (Electron_State(:,1)>80e-9 &
        ↪  Electron_State(:,1)<120e-9...
156     & ((Electron_State(:,2)>=60e-9 & Electron_Prev_State(:,2)>60e-9)) &
        ↪  Electron_Prev_State(:,1)<80e-9),1);
157 %Region 4
158 Electron_State( ( (Electron_State(:,1)>80e-9 & Electron_State(:,1)<120e-9 )&
    ↪  ((Electron_State(:,2)>60e-9 & Electron_Prev_State(:,2)>=60e-9)) &...
159     Electron_Prev_State(:,1)>120e-9),1) = 2*120e-9 - Electron_State( (Electron_State(:,1)>80e-9 &
        ↪  Electron_State(:,1)<120e-9 &...
160     ((Electron_State(:,2)>60e-9 & Electron_Prev_State(:,2)>=60e-9)) &
        ↪  Electron_Prev_State(:,1)>120e-9),1);
161
162 %Top Horizontal
163 Electron_State( (Electron_State(:,1)<120e-9 & Electron_State(:,1)>80e-9 &
    ↪  (Electron_State(:,2)<40e-9 & Electron_Prev_State(:,2)>=40e-9) &...
164     (Electron_State(:,4)<0)),2) =  2*40e-9 + Electron_State( (Electron_State(:,1)<120e-9 &
        ↪  Electron_State(:,1)>80e-9 &...
165     (Electron_State(:,2)<=40e-9 & Electron_Prev_State(:,2)>=40e-9) & (Electron_State(:,4)<0)),2);
166 %Bottom Horizontal
167 Electron_State( (Electron_State(:,1)<120e-9 & Electron_State(:,1)>80e-9 &
    ↪  (Electron_State(:,2)>60e-9 & Electron_Prev_State(:,2)<=60) &...
168     (Electron_State(:,4)>0)),2) =  2*60e-9 - Electron_State( (Electron_State(:,1)<120e-9 &
        ↪  Electron_State(:,1)>80e-9 &...
169     (Electron_State(:,2)>=60e-9 & Electron_Prev_State(:,2)<=60) &(Electron_State(:,4)>0)),2);
170 %Electron shifting end
```

Source Code 12: Box boundary - Electron shifting

The change in electron position is relative to where it bounces off, therefore in the matrix indexing the condition was set to determine its the difference between the previous position and the current position.

3.3 Plotting density and temperature map

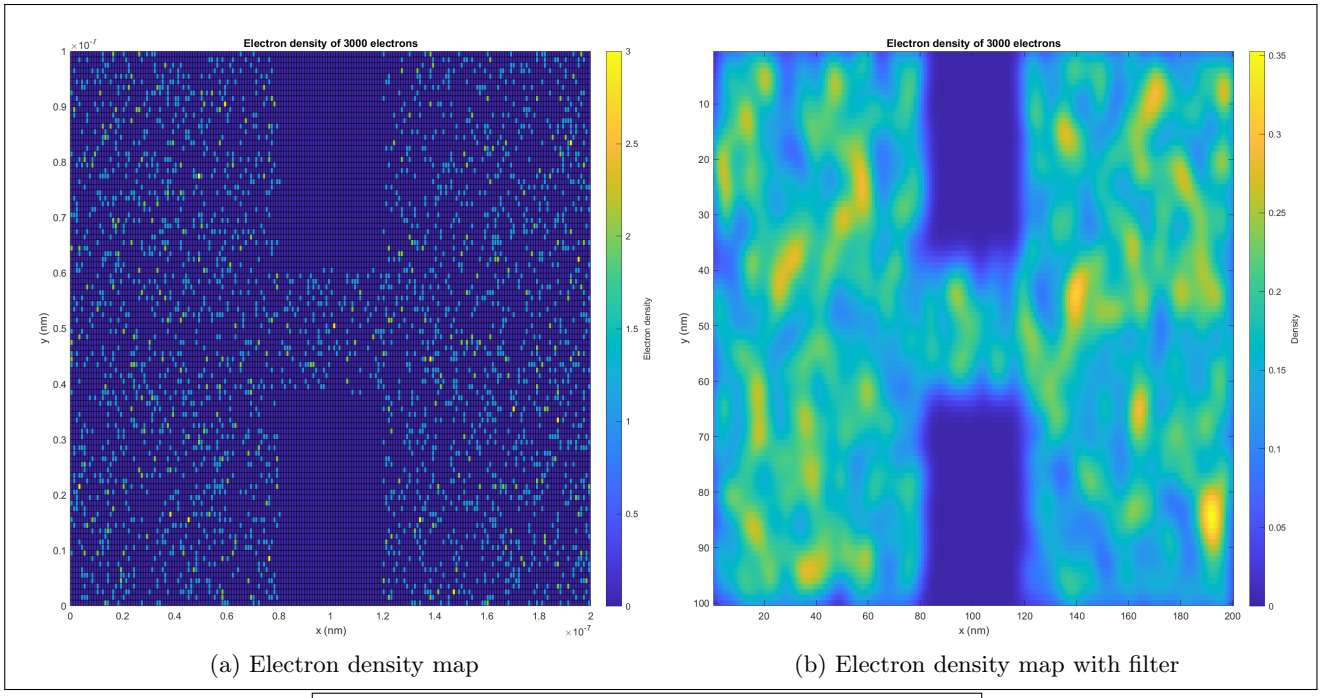
The final section of the code plotted the temperature and density map from the simulation data. The density map was simple and implemented using the **hist3** function, while the temperature plot was done via binning. Additionally, to smooth out the data a Gaussian filter was applied[1] to both the density and temperature maps. Source code 13 below shows how this mapping was implemented.

```
figure("name","Electron Density")
X = [Electron_State(:,1) Electron_State(:,2)];
hist3(X,'Nbins',[200 100],'CdataMode','auto')
axis([0 Length 0 Height])
xlabel('x (nm)')
ylabel('y (nm)')
colormap;
c = colorbar;
c.Label.String = 'Electron density';
view(2)
title(sprintf("Electron density of %d electrons",nElectrons));
%With Gaussian filtering :
Density = hist3(Electron_State(:,1:2),[200 100])';
N = 20;
sigma = 3;
%Creating a Gaussian filtering matrix
[x,y]=meshgrid(round(-N/2):round(N/2), round(-N/2):round(N/2));
F=exp(-x.^2/(2*sigma^2)-y.^2/(2*sigma^2));
F=F./sum(F(:));
figure("name","Electron Density (with Gaussian filtering)")
imagesc(conv2(Density,F,'same'))
xlabel('x (nm)')
ylabel('y (nm)')
c = colorbar;
c.Label.String = 'Density';
view(2)
title(sprintf("Electron density of %d electrons",nElectrons));
%For the temperature density plot the electrons positions were binned
Temperature_Sum_X = zeros(ceil(Length/1e-9),ceil(Height/1e-9));
Temperature_Sum_Y = zeros(ceil(Length/1e-9),ceil(Height/1e-9));
Temperature_Bin_Count = zeros(ceil(Length/1e-9),ceil(Height/1e-9));
% Look at velocities of all the particles
for i=1:nElectrons
    % Find which "bin" it belongs in:
    x = floor(Electron_State(i,1)./1e-9);
    y = floor(Electron_State(i,2)./1e-9);
    if(x==0)
        x = 1;
    end
    if(y==0)
        y= 1;
    end
    % Add its velocity components to the cumulative count:
    Temperature_Sum_X(x,y) = Temperature_Sum_X(x,y) + Electron_State(i,4)^2;
    Temperature_Sum_Y(x,y) = Temperature_Sum_Y(x,y) + Electron_State(i,3)^2;
    Temperature_Bin_Count(x,y) = Temperature_Bin_Count(x,y) + 1;
end
% Now, with the velocities added up, calculate the temperatures:
TemperatureDensity = (Temperature_Sum_X + Temperature_Sum_Y).*Mass_n./k./2./Temperature_Bin_Count;
%If somewhere in the calculation the density beame nan then make it 0
TemperatureDensity(isnan(TemperatureDensity)) = 0;
%Transpose the matrix
TemperatureDensity = TemperatureDensity';
figure("Name","electron heat density (with Gaussian filtering)");
imagesc(conv2(TemperatureDensity,F,'same'));
set(gca,'YDir','normal');
title('Temperature Map');
c = colorbar;
c.Label.String = 'Temperature (K)';
xlabel('x (nm)');
ylabel('y (nm)');
```

Source Code 13: Electron density and temperature mapping

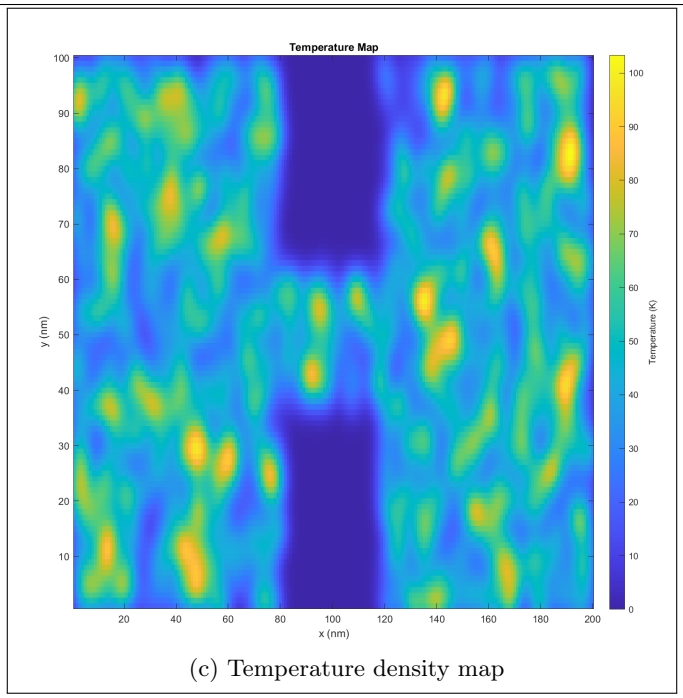
3.4 Part 3 Simulation results

Figure 4a below shows the results of the density mapping, and Figure 4b shows the same data but filtered, Figure 4c shows the filtered temperature map. This data was collected using scattering enabled and all boundaries specular.



(a) Electron density map

(b) Electron density map with filter



(c) Temperature density map

Figure 4: Mapping the simulation results from Part 3

4 Reference

[1] gnovice, "Matlab - Creating a heatmap to visualize density of 2D point data," Stack Overflow, 17-Nov-2017. [Online]. Available: https://stackoverflow.com/questions/46996206/matlab-creating-a-heatmap-to-visualize-density-of-2d-point-data. [Accessed: 07-Feb-2021].