ECE 2036 Fall 2017

Lab 6: Electron Drift in Silicon

Assigned: Nov 17, 2017 *Due: 11:59 PM Dec 1, 2017*

In this lab, you will write code to simulate the (2d) path of an electron moving through silicon due to an applied electric field. As you know, an electron is accelerated by an applied electric field. However, when an electron is traveling through a material like silicon, it is scattered at random intervals by atomic lattice vibrations, lattice defects, other electrons, and surfaces. We will use the Drude model. According to this model, after a scattering event the electron's velocity is randomized (with the new average velocity depending on temperature) and has no relationship to its velocity before it was scattered. Therefore, each electron will follow a random path through the semiconductor. You will simulate many of these random paths and average the results to find the 'average' behavior of an electron. Finally, you will write code to farm out the simulations to different cores on the deepthought cluster.

Equations of motion

In the classical picture, an electron obeys Newton's equations of motion. Assuming an electric field *E* applied in the *z* direction:

$$\vec{F} = m^*\vec{a} = -eE\hat{z}$$

where *F* is the force, $m^* = 1.12\ m_0$ is the effective mass of the electron in silicon, *a* is the acceleration, and *e* is the charge of an electron. ($m_0 = 9.109e\text{-}31\ kg;\ e = 1.602e\text{-}19\ C$). We will be solving for the velocity of the electron as a function of time, so the relevant equation for our purposes is:

$$\frac{d\vec{v}}{dt} = \vec{a} = \frac{-eE\hat{z}}{m^*}$$

To keep things (relatively) simple, we'll only consider the *x* and *z* components of velocity:

$$\frac{dv_z}{dt} = \frac{-eE}{m^*}$$

$$\frac{dv_x}{dt} = 0$$

We can write closed-form solutions to each of these equations:

$$v_z = v_{z,initial} - \frac{eE}{m^*}t$$

$$v_x = v_{x,initial}$$

However, this does not account for the scattering events that were mentioned in the introduction. The scattering events occur at random intervals, and each scattering event randomizes the velocity of the electron. We know that the average time between scattering events at room temperature is given by $\tau = 8.92e\text{-}13\ s$. This is referred to as the scattering time. There's more than one way to deal with scattering, but our approach will be to divide time up into tiny discrete steps of duration Δt, which are much smaller than τ. We will loop over time steps, and on each time step, we'll 'roll the dice' to determine whether a scattering event occurred in that time step. The probability per unit time of a scattering event is $(1 / \tau)$, so the probability of a scattering event occurring in any given time step is $P = (\Delta t / \tau)$. Therefore, on each time step, you should generate a random number between 0 and 1. If your random number is less than P, that means a scattering event occurred in that time step; otherwise a scattering event did not occur. What we do next depends on the outcome:

- If a scattering event occurred, we'll randomize the electron's velocity. In principle, we should randomize both the magnitude and direction of the velocity, but for the purposes of this lab we'll just randomize the direction. You'll have to generate a random angle θ between 0 and 2π representing the new angle of the electron velocity. Set $v_z = v_{th} \cos\theta$ and $v_x = v_{th} \sin\theta$, where the magnitude v_{th} is given by
$$v_{th} = \sqrt{\frac{2\,kT}{m^*}}$$
- If a scattering event didn't occur, we'll solve the equations of motion given above. If the velocity after the i^{th} time step was $(v_{x,i}\,,\ v_{z,i}\,)$ and the new velocity is $(v_{x,i+1}\,,\ v_{z,i+1}\,)$, we calculate the new velocity from the old using $v_{x,i+1} = v_{x,i}$ and
$$v_{z,i+1} = v_{z,i} - \frac{eE}{m^*}\Delta t$$

In this way you will fill arrays holding the x and z components of the velocity at each time step.

We would also like to determine the position of the electron at each time step. After you have calculated the velocity at each time step, you can go back and use the following equations of motion to determine the position:

$$\frac{dx}{dt} = v_x$$

$$\frac{dz}{dt} = v_z$$

You should use a similar procedure as the one outlined above, but you don't have to worry about the scattering events this time around, because they were already accounted for when you calculated the velocity. Assume that $(x, z) = (0, 0)$ at $t = 0$, and find (x_{i+1}, z_{i+1}) using

$$x_{i+1} = x_i + v_{x,i}\Delta t$$
$$z_{i+1} = z_i + v_{z,i}\Delta t$$

Multithreading

As you know, most modern computer microprocessors have multiple cores. (It has proven more practical to increase the number of cores rather than increase the clock speed further, which causes excess heating.) The deepthought cluster is composed of many microprocessors linked together, each of which has many cores. This parallel computing power does not automatically accrue benefits to our programs, however. In order to exploit the parallel processing capabilities of a cluster like deepthought, we must write code that splits up its tasks into multiple threads, each of which can be performed on a separate core. Some problems are more amenable to multithreading, and others are less so. We will exploit multithreading by calculating different electron paths using different cores.

Lab Programming Instructions

You must download two files from t-square: gthread.cc and gthread.h. Do not modify these files. They were written by Dr. George Riley to provide easy-to-use multithreading functionality, and their use will be described in class. For the purposes of this lab, you will need to use three functions provided by gthread: `CreateThread`, `EndThread`, and `WaitAllThreads`.

Connect to deepthought and create a subdirectory named 'Lab6' in your home directory. You should transfer gthread.cc and gthread.h to the Lab6 subdirectory. Inside the Lab6 subdirectory, create a new file called electron_sim.cc. You will write your code in electron_sim.cc. The correct command to use when compiling your code on deepthought is as follows:

```
g++ -std=c++0x electron_sim.cc gthread.cc -lpthread
```

Be sure to `#include "gthread.h"` at the top of your electron_sim.cc file.

Detailed instructions for your electron_sim.cc code are as follows:

- define some fundamental global constants that you will need, such as the Boltzmann constant k, etc.
- write a class called `electronPath,` which should include the following data and function members:
    - data members to hold the effective mass m^*, electric field E, scattering time τ, total simulation time, and time step size Δt. These values should all be passed in via a constructor.
    - data member to hold the total number of time steps
    - data member dynamically allocated arrays to hold the time t, the x and z positions, and the velocity components v_x and v_z on each time step
    - constructor with five arguments, which are copied into the data members described above. This constructor should perform dynamic allocation for the time, position, and velocity arrays, and fill the time array.
    - copy constructor, assignment operator, and a destructor (rule of three)
    - `simulateVelocity` member function to calculate the velocity components v_x and v_z on each time step, and store them in the velocity arrays.
    - `simulatePosition` member function to calculate the position x and z on each time step, and store them in the position arrays.
    - `printToFile` member function to print an `electronPath`'s time t and x and z positions on each time step to a data file in comma-separated format.
    - overloaded addition operator to add two `electronPath`s (which will be used to average many paths in a different function)
    - overloaded division operator to divide an `electronPath` by a `double` (which will also be used to average many paths in a different function)
- write a standalone function `run_sim`, which returns nothing, and takes a pointer to an `electronPath` as its sole argument. It should call `simulateVelocity` and `simulatePosition` on its `electronPath` pointer argument.
- write a standalone function `average_paths`, which returns an `electronPath`, and takes a vector of `electronPath`s as its sole argument. It should return a new `electronPath` which holds the average of all the `electronPath`s in the vector.
- your main function should prompt the user for the number of `electronPath`s to be simulated, and store the `electronPath`s in a vector. It should call `run_sim` for each `electronPath` and then call `average_paths` to average all of the calculated `electronPath`s.
- you should use the gthread functions to 'farm out' electron path calculations to different cores. The exact method you use to do this is up to you.

All of your `electronPath`s should use the following parameters:

E = -1.0e5 *V/m*; total simulation time = 1000*τ; 1,000,000 time steps

When your code is complete, you should run your code for 100 electron paths. Your code should output one data file representing a single `electronPath` of your choice, and another data file representing the averaged `electronPath`. Import these data files into Matlab. For each data file, generate a plot of z vs. x and z vs. t. For the averaged `electronPath`, calculate the average z-velocity of the electron's path by dividing the maximum value of z by the maximum value of t, and compare it to the following equation:

$$v_d = \mu E = \frac{e\,\tau\,E}{m^*}$$

where v_d is the electron drift velocity, and μ is the electron mobility. Use Matlab to print the four figures described above to .png files, and upload those files to deepthought. The four png files should be named path_zvx.png, path_zvt.png, avgpath_zvx.png, and avgpath_zvt.png.

Turn-in Procedures

You must submit your source code and png figure files on the deepthought cluster by using the turnin script, as you did for Lab 0. You MUST ensure that your code compiles and runs properly on deepthought.

Depending on your section, from your home directory enter one of the following at the command prompt:

turnin-ece2036a Lab6

or

turnin-ece2036b Lab6

This automatically copies everything in your Lab6 directory to a place that we can access (and grade) it. Your Lab6 directory should include your source code as well as the png figure files described above.

Grading Rubric

GRADING POINT DEDUCTIONS

Element	Percentage Deduction	Details
Does not compile	30%	Does not compile on deepthought.
electronPath does not include required function members	30%	Self-explanatory
Output figures not present / do not match expectations	20%	
Does not use multithreading	10%	
Does not use vector properly	10%	Does not use vector as required

LATE POLICY

Element	Percentage Deduction	Details
First Day	20%	By Monday after the due date
Each Additional Day	20% per day	The weekend (Sat/Sun) will count as one day