Report

1.Introduction

I have written a program to simulate a velocity filter. The motivation behind building velocity filters are many applications that require particles of a certain velocity. Example uses include accelerator mass spectrometry.

2. Methods

The only force affecting the particles inside the box is the Lorentz force (1),

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times B). \quad (1)$$

From this equation we can derive the optimal velocity for the particle

$$v=E/B. \quad (2)$$

This is the particle velocity that will result in a straight line of motion, without acceleration.

In the simulation, the particle movement is integrated with the Euler method. For the current step (xi , vi) we calculate acceleration, velocity and position in the following way:

$$a_i = F(v_t)/m \quad (2)$$

$$v_t+\Delta t = v_t+a_t\Delta t \quad (3)$$

$$x_t+\Delta t = x_t + v_{t+\Delta t}\,\Delta t \quad (4)$$

3.Implementation of methods

The methods described in section 2 are implemented in a straight forward manner in the program. Two derived data types are used in the program, particle and vector. Calculation of the force (1) and acceleration (2) are done in the main program, since they require knowledge of the fields and the structure of the box. Velocity and location updates (3)(4) are handled inside the particle module.

3.1 Overview of structure

This section aims to give an overview of the program and it’s operation.

Figure 1 shows all the different modules and programs, and how they relate to each other.

As we see in the figure 1, main program uses the inputfile handler to read the file and create the particles. Main also uses cmd_line module to read in the command line arguments. With the arguments we create a time step and the two field vectors that the main uses, electric and magnetic.
The procedural steps taken after the program starts are more carefully displayed in the sequence diagram of figure 2.

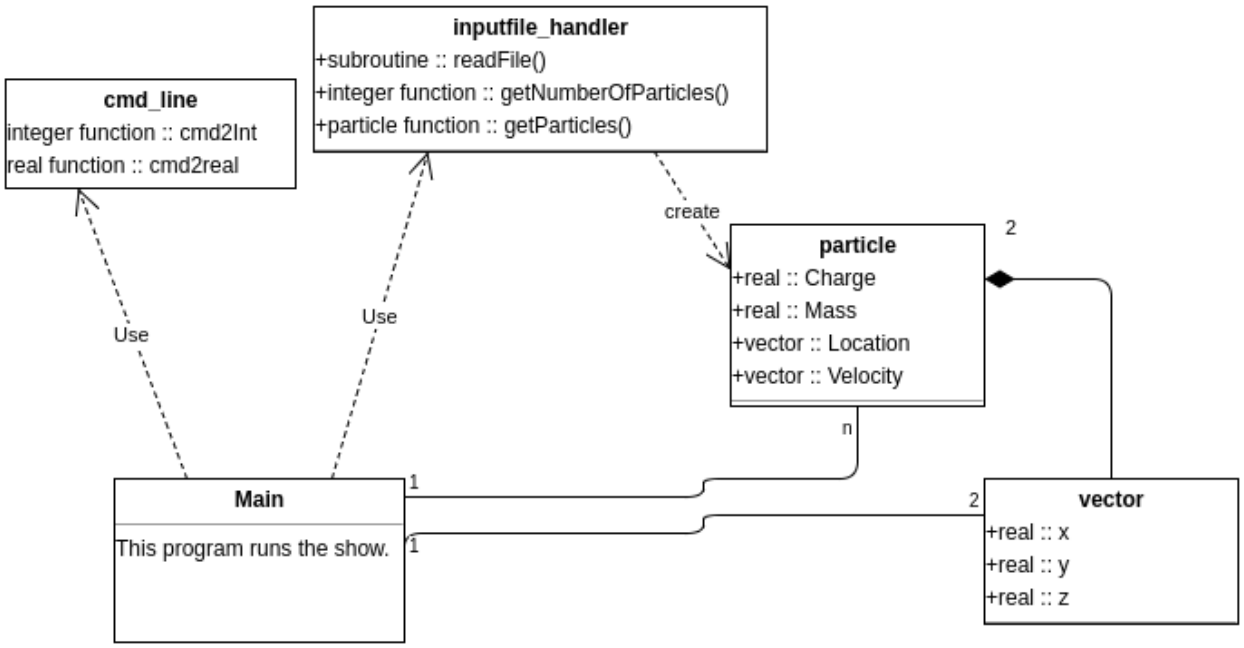


Figure 1

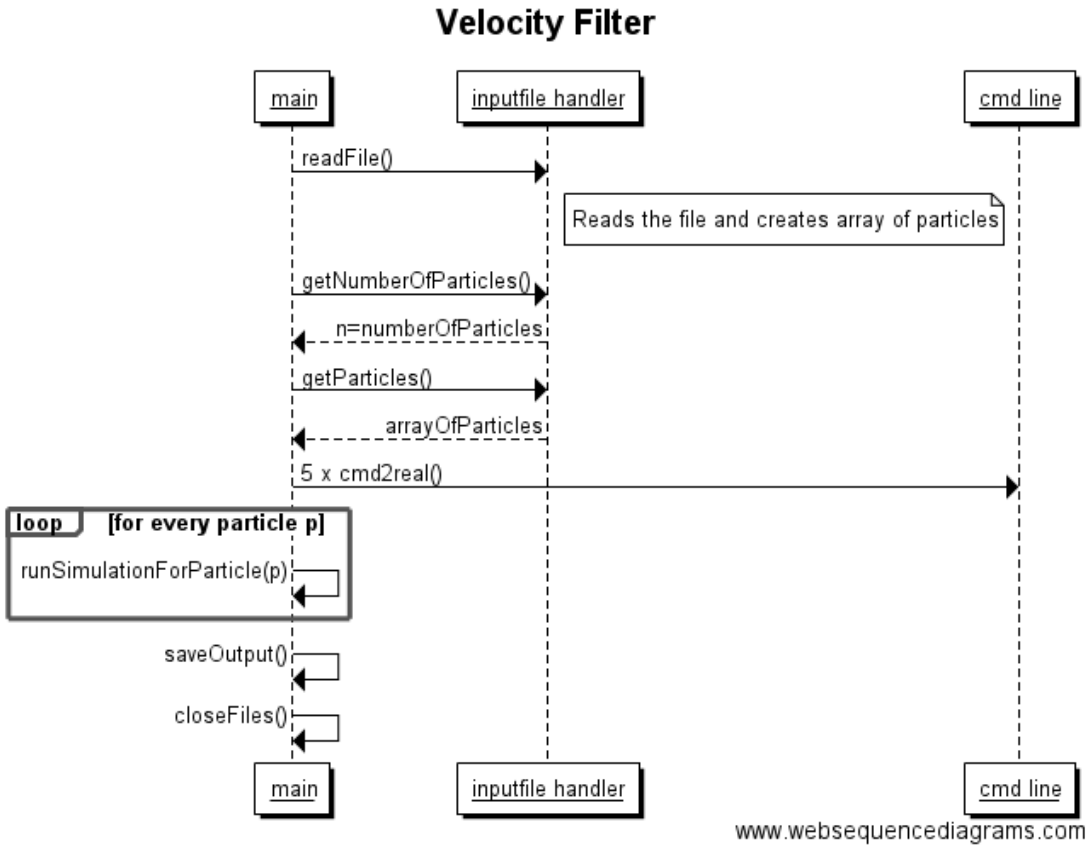


Figure 2

For detailed info about how the functions and subroutines work, please see the source code and read the comments.

3.2 Coordinate system

The problem situation is treated by the program as follows.

entry point of each particle is fixed to purple dot in figure 3.
origin of the coordinate system is set to red dot in figure 3.
axis directions are as displayed in the figure 3.

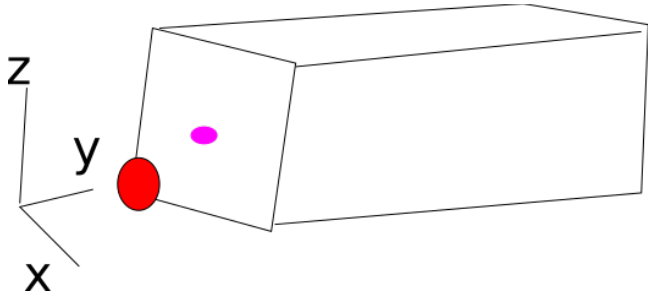


Figure 3

3.3 Input and output

3.3.1Units

The units used in inputs should be the following kinds:

- Coulombs for the charge
- Kilograms for mass
- Meters per second for velocities
- Tesla's for magnetic field strengths.
- Volts per meter for electric field strengths
- Seconds for time step
- Meters for distances

Output follows similar conventions.

3.3.2 Input

There are two kinds of input that the program requires.
1. Input from a file (‘input.txt’)
2. Input from the command line

1. The input file should contain all the information about the particles to be simulated.
The first line should contain the number of particles in the file, nothing more.
Rest of the lines are arranged as in the figure 4 one line for one particle in the following manner:

particle identifier, charge, mass, initial velocity in X direction, initial velocity in Y direction, initial

velocity in Z direction.

```
6
1,-1.6021766208E-19,9.10938356E-31,0,0.12E6,0
2,-1.6021766208E-19,1.672621898E-27,0,0.12002E6,0
3,-1.6021766208E-19,1.672621898E-27,0,0.14E6,0
4,-1.6021766208E-19,1.672621898E-27,0,0.10E6,0
5,-1.6021766208E-19,1.672621898E-27,-0.1E5,0.7E6,0.1E5
6,-1.6021766208E-19,1.672621898E-27,0,0.17E6,0
```

Figure 4

2. When the program is run, it should be given the following parameters via command line(in this order) :

timestep to be used, electric field x component, -y component, z- component, magnetic field x component, -y component, z-component.

3.3.3 Output

There are two kinds of output that the program creates.
1. A file (‘output.txt’) containing data from particles that made it through the box.
2. A file (‘trajectories.xyz’) containing trajectories of ALL particles simulated.

1. The format of the data displayed in output.txt file is seen in figure 5.

```
    4
id|    charge|      mass|        vx|        vy|        vz|         x|         y|         z|
 1|-.1602E-18|0.9109E-30|0.0000E+00|0.1200E+06|0.0000E+00|0.9500E-02|0.7600E-01|0.9500E-02|
 2|-.1602E-18|0.1673E-26|0.0000E+00|0.1200E+06|-.3005E+02|0.9500E-02|0.7600E-01|0.9500E-02|
 3|-.1602E-18|0.1673E-26|0.0000E+00|0.1397E+06|-.3695E+04|0.9500E-02|0.7600E-01|0.9535E-02|
 4|-.1602E-18|0.1673E-26|0.0000E+00|0.1007E+06|0.5134E+04|0.9500E-02|0.7600E-01|0.9431E-02|
```

FIgure 5

2. The trajectory file is a .xyz file. First line tells how many rows/frames there are about this particular particle. The following lines are info about the location of the particle. They are displayed in chronological order. As seen in figure 6, the first column is the particle identifier, second is position on x-axis, third on y-axis and fourth on z-axis.

```
6335
#id                x                y                z
 1          0.950000E-02     0.120000E-05     0.950000E-02
 1          0.950000E-02     0.132000E-04     0.950000E-02
 1          0.950000E-02     0.252000E-04     0.950000E-02
 1          0.950000E-02     0.372000E-04     0.950000E-02
```

Figure 6

NOTE. I have put all the trajectories to the same file, contrary to the instructions to create them separately. Ovito can nevertheless display all of these trajectories once the ‘File contains time series’ box is ticked.

3.4 Running the program

The program can be compiled and run with the following commands:

To compile the program:
gfortran -c vector.f90 && gfortran -c particle.f90 && gfortran -c inputfile_handler.f90 && gfortran -c command_line_argument_module.f90 && gfortran -c vector.f90 && gfortran -c main.f90 && gfortran main.o vector.o particle.o command_line_argument_module.o inputfile_handler.o

To run the program:
./a.out 0.00000000001 0 0 12000 0.1 0 0

4 Results

The program has been tested with three kinds of particles. In this document, only three trajectories (of protons and electrons) produced by the tests have been visualized with Ovito. The following parameters were used in all the tests, a time step of 0.1 picoseconds, electric field (0, 0, 12kV), magnetic field (0.1, 0, 0).

First trajectory is displayed in figure 7. This trajectory belongs to an electron entering the box with initial velocity of 1.2*10^5 m/s in the y-direction. As we can see from figure 7 the trajectory is a straight line.

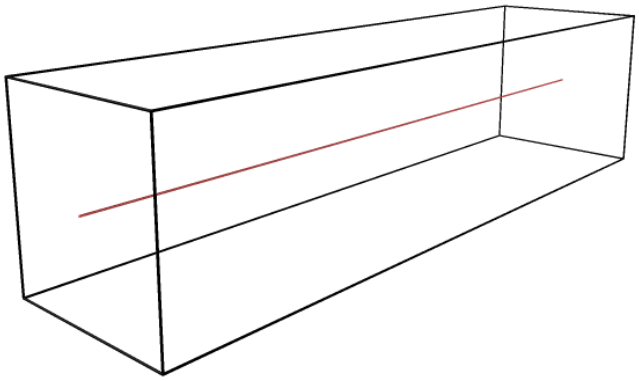


figure 7

This is no coincidence. The initial velocity is the critical velocity $v=E/B$ (2)

*W*e conclude that here, the program provides a trajectory supported by the theory.

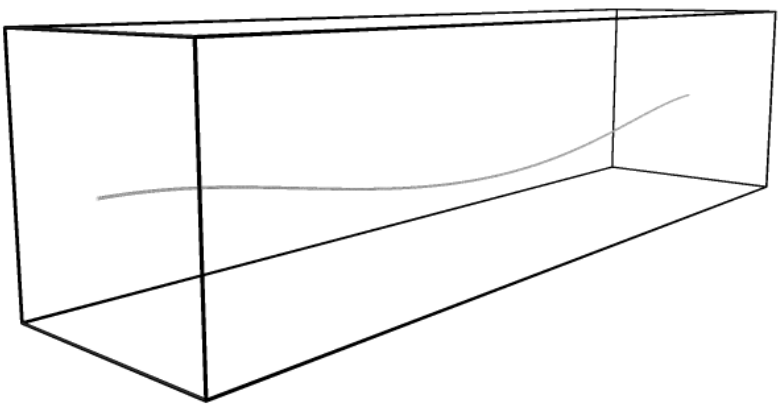
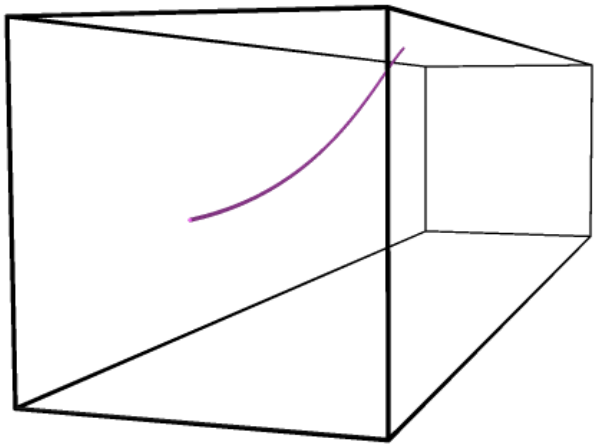


figure 8

The second trajectory is displayed in figure 8. This figure shows a proton entering the box with a initial velocity of 1.0*10^5 m/s in the y-direction.

We can see that this is no straight line. The initial velocity is pretty far from the critical v=E/B velocity, but the proton is massive enough not to get thrown out of the box by the forces.

Third trajectory is displayed in figure 9. This figure shows a proton, coming in too fast with a initial velocity of 1.7*10^5 m/s in the y-direction.



The protons trajectory is bent too much in the z-direction and it hits the roof of the box.

figure 9

5 Conclusions

As discussed, the program currently uses the Euler method to integrate the particles movement. This is a first order approximation and the accuracy of the program could easily be improved by replacing the method with something more sophisticated.