بسم الله الرحمن الرحيم

AECENAR
Association for Economical and Technological Cooperation in the Euro-Asian and North-African Region
www.aecenar.com

IAP
INSTITUTE FOR ASTROPHYSICS

# IAP-PSC
# Plasma Simulation Code
# (Particle-in-Cell Code)

Authors:
Maryam ABDEL-KARIM
Editor: Samir Mourad

Last Update: 23.08.2019 14:26

D:\AECENAR\IAP\IAP-TheoreticalAstrophysics\IAP-PSC\230819_14IAP-PSC.docx

## Content

**Preface** ........ **1**

**1 Basics** ........ **3**

1.1 Laser-matter interaction ........ 3

1.1.1 Plasma definition ........ 3

1.1.2 Laser-cutting ........ 4

**2 Plasma Waves Kinetic Theory** ........ **6**

2.1 Governing Equations, Vlasov-Boltzmann equation ........ 6

2.1.1 Transport equations, Maxwell Equations ........ 6

2.1.1.1 The Boltzmann collision operator ........ 7

**3 Numerical model** ........ **9**

3.1 Meshing ........ 9

**4 Some public codes / research groups** ........ **11**

4.1 The Plasma Theory and Simulation Group PTSG Univ. of California, Berkley ........ 11

4.1.1 People of PTSG ........ 12

4.1.2 Projects ........ 12

4.1.2.1 Current Projects in PTSG ........ 12

4.1.2.2 Recently Completed Projects in PTSG ........ 12

4.1.3 PTSG Software ........ 12

• General Information ........ 12

• Acknowledgments ........ 12

• Description ........ 13

• Distribution ........ 13

Codes that require xgrafix ........ 14

Codes with own version of xgrafix ........ 14

Python/Matlab based codes ........ 14

Code Consulting and Maintanence ........ 14

4.1.4 Publications Using PTSG Codes, Worldwide ........ 14

4.1.5 Workshops ........ 15

4.1.5.1 Plasma Device Workshop 2009 (PDW2009) ........ 15

**5 Comparison between actual IAP-PCS and XPDP2** ........ **17**

5.1 Description of XPDP2 ........ 17

5.1.1 Abstract of [1]: ........ 17

5.1.2 Abstract of [2]: 17
5.1.3 xpdp2 Code Description 18
**6 IAP-PSC Program 23**
6.1 Description 23
6.2 The Code 24
6.2.1 Class 1: parameters 24
6.2.2 Class 2: Lorentz 26
6.2.3 Class 3: pos_veloct 26
6.2.4 Class 4: Dens_current 27
6.2.5 Class 5: Maxwell_equat 27
6.2.6 Main program 29
6.3 Results 35
**Literature 37**

# Preface

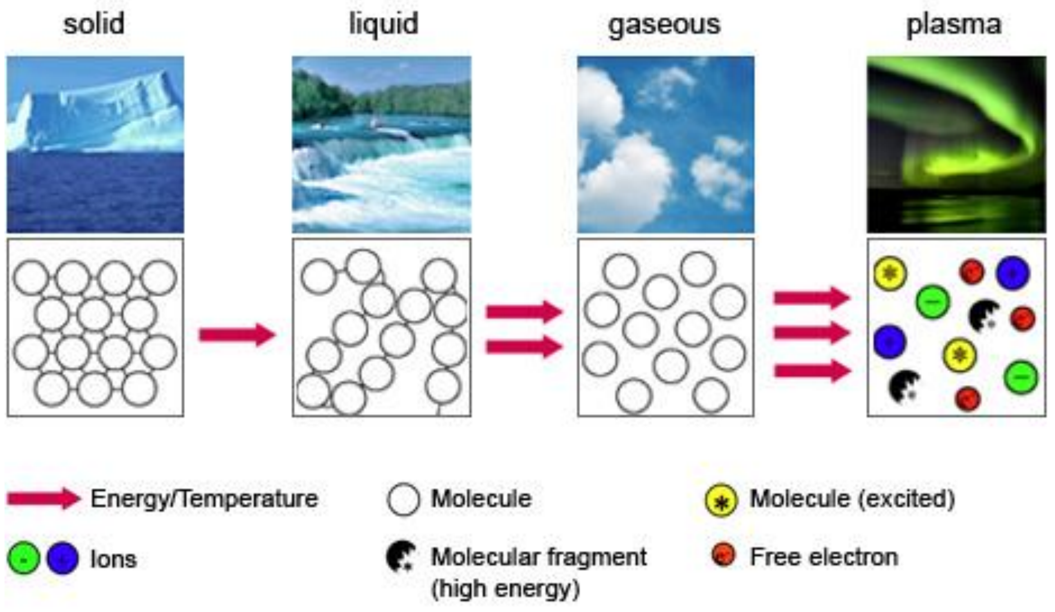
Text

# 1 Basics

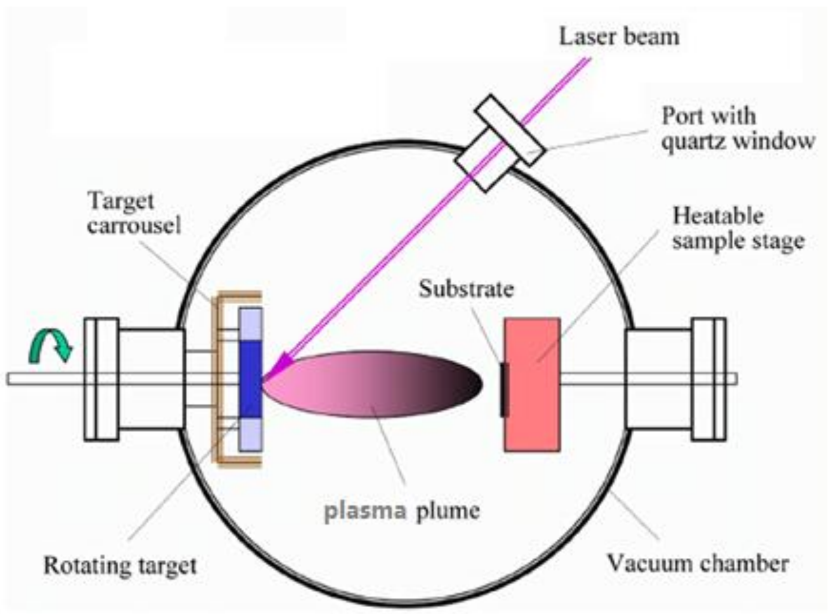
## 1.1 Laser-matter interaction

### 1.1.1 Plasma definition

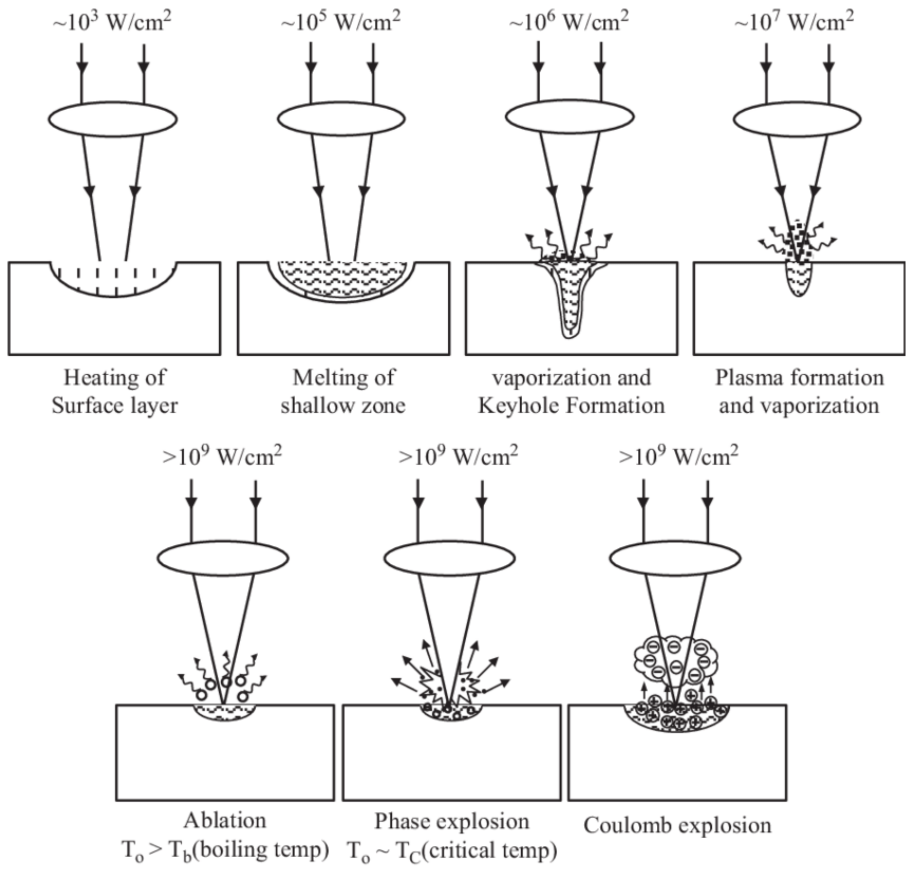
Plasma is considered the fourth state of matter. The three other states are solid, liquid, and gas. Plasma is a cloud of protons, neutrons and electrons where all the electrons have come loose from their respective molecules and atoms, giving the plasma the ability to act as a whole rather than as a bunch of atoms. A plasma is more like a gas than any of the other states of matter because the atoms are not in constant contact with each other, but it behaves differently from a gas. It has what scientists call collective behavior. This means that the plasma can flow like a liquid or it can contain areas that are like clumps of atoms sticking together.



Plasma results from the interaction between a laser beam and a metal target (see figure below).



The power of the laser used affect the phenomena that will occur.



### 1.1.2 Laser-cutting

There are many different methods in cutting using lasers, with different types used to cut different material. Some of the methods are vaporization, melt and blow, etc.

Vaporization:
Laser heats surface to vaporization
Forms keyhole
Now light highly absorbed in hole (light reflects until absorbed)
Vapor from boiling stabilizes molten walls
Material ejected from hole can form Dross at bottom and top
In materials that do not melt, just vapor escapes

e.g. Wood, carbon, some plastics

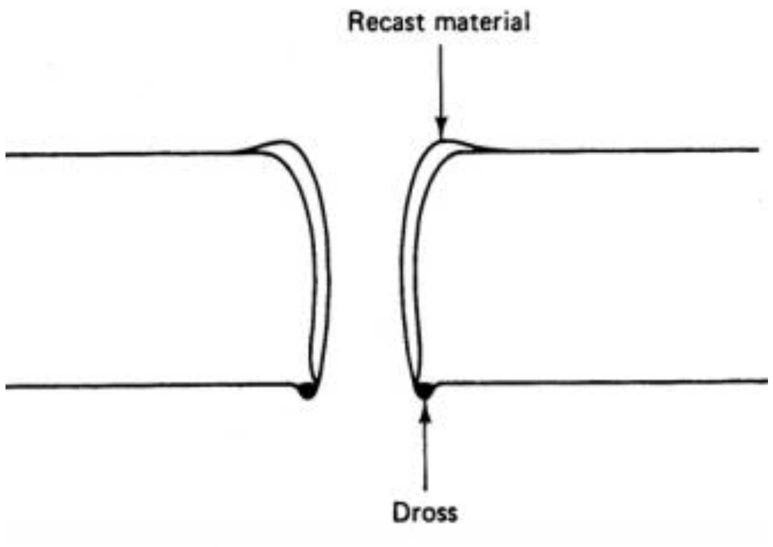


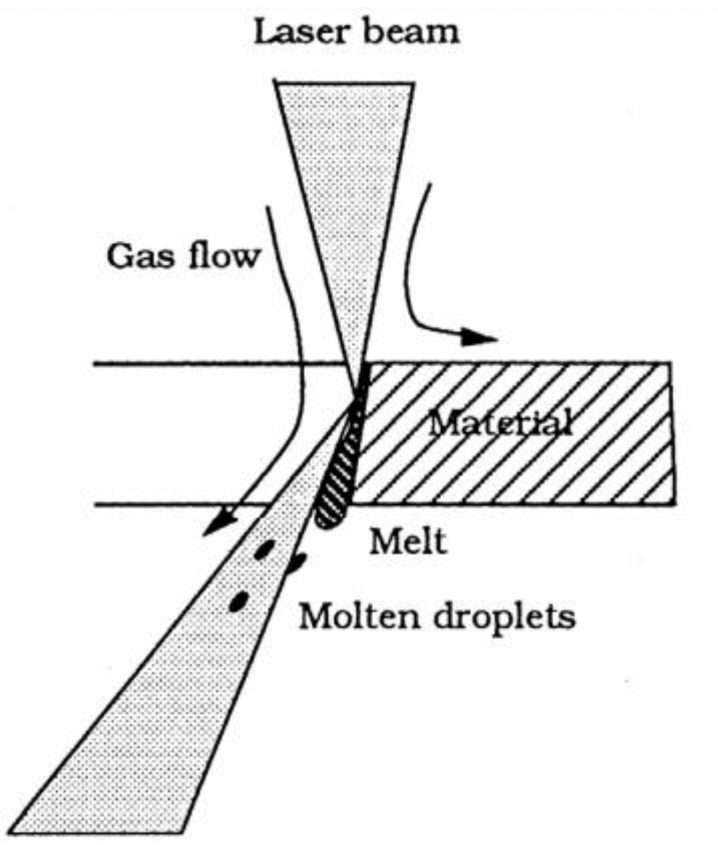
**Figure 13-21** Sketch of laser-drilled hole.

Melt and Blow

Once melt is formed use gas flow to blow away materials

Do not need to vaporize, thus power reduced by factor of about 10

e.g. Metals.



# 2 Plasma Waves Kinetic Theory

## 2.1 Governing Equations, Vlasov-Boltzmann equation

Intense laser radiation interacting with matter at relativistic intensities (a > 1) is capable of generating plasma far away from equilibrium. A promising approach to describe the nonlinear, kinetic nature of the interaction are plasma models based on the fully relativistic Vlasov-Boltzmann equations combined with Maxwell's equations.

So, the governing equations of intense laser plasma interaction, are the Vlasov-Boltzmann equations combined with Maxwell's equations in three spatial and momentum dimensions.

The Vlasov equation is a differential equation describing time evolution of the distribution function of plasma consisting of charged particles with long-range interaction.

We consider a plasma consisting of electrons and ions, which are represented by distribution functions $f_k$ ($\vec{x}$, $\vec{p}$, t). The distribution functions $f_k$ give the probability of finding particles of sort k in a given volume of phase space. We assume that the electrons and ions in the plasma under consideration interact via electromagnetic radiation and binary collisions. Hence, an appropriate description of the plasma is based on the following set of transport equations, which for brevity are stated in covariant form.

### 2.1.1 Transport equations, Maxwell Equations

$$p_k^\mu \frac{\partial f_k}{\partial x^\mu} + m_k F_k^\mu \frac{\partial f_k}{\partial p_k^\mu} = \sum_{l=n,e,i} \int \frac{d^3 p_l}{p_l^0} F_{kl} \int d\Omega_\psi \, \sigma^{kl}(s,\psi) \left( f_k' f_l' - f_k f_l \right)$$

The quantity $d\Omega_\Psi$ is an element of solid angle between $\overrightarrow{p'}_k$ and $\vec{p}_k$ and $\sigma^{kl}(s,\Psi)$ denotes the invariant cross section. The relative velocity between particles k and l is given by

$$v_{kl} = \frac{cF_{kl}}{p_k^0 p_l^0} = \sqrt{(\vec{v}_k - \vec{v}_l)^2 - \frac{1}{c^2}(\vec{v}_k \times \vec{v}_l)^2} \, .$$

The force on the charged particles in the plasma is the Lorentz force given by

$$F_k^\mu = \frac{q_k}{m_k c} F^{\mu\nu} p_{k\nu} \, .$$

Maxwell's equations are represented as:

$$\frac{\partial}{\partial x^\mu} F^{\mu\nu} = \frac{j^\nu}{c\epsilon_0} \, , \qquad \frac{\partial}{\partial x^\mu} \tilde{F}^{\mu\nu} = 0$$

$$j^\nu = \sum_{k=n,e,i} q_k \int d^4 p \, 2\Theta(p_0) \, \delta(p^2 - m_k^2 c^2) \, c p^\nu \, f_k$$

Equivalently, Eqn. (3.1) can be rewritten in three vector notation

$$\left(\partial_t + \vec{v}_k \partial_{\vec{x}} + q_k \left[\vec{E} + \vec{v}_k \times \vec{B}\right] \partial_{\vec{p}_k}\right) f_k$$
$$= \sum_{l=n,e,i} \int d^3p_l \, v_{kl} \int d\Omega_\psi \, \sigma^{kl}(s,\psi) \left(f_k^{'} f_l^{'} - f_k f_l\right) .$$

Maxwell equations become

$$\begin{aligned}
\partial_t \vec{E} &= c^2 \vec{\nabla} \times \vec{B} - \vec{j}/\epsilon_0 , \\
\partial_t \vec{B} &= -\vec{\nabla} \times \vec{E} , \\
\partial_t \rho &= -\vec{\nabla} \cdot \vec{j} .
\end{aligned}$$

The charge and current densities in three notations are given by

$$\begin{aligned}
\rho &= q_e \int d^3p_e f_e + q_i \int d^3p_i f_i , \\
\vec{j} &= q_e \int d^3p_e \vec{v}_e f_e + q_i \int d^3p_i \vec{v}_i f_i .
\end{aligned}$$

### 2.1.1.1 The Boltzmann collision operator

$$C_{kl} = \int \frac{d^3p_l}{p_l^0} \int \frac{d^3p_k^{'}}{p_k^{0\,'}} \int \frac{d^3p_l^{'}}{p_l^{0\,'}} \left(W_{klk\,'l\,'} \, f_k^{'} f_l^{'} - W_{k\,'l\,'kl} \, f_k f_l\right)$$

$$\begin{aligned}
W_{k\,'l\,'kl} &= \frac{e_k^2 e_l^2 m_k^2 m_l^2 c^2}{4\pi^2 \epsilon_0^2} \left|M_{k\,'l\,'kl}\right|^2 \delta^4\left(p_k^{'} + p_l^{'} - p_k - p_l\right) \\
&= s\,\sigma^{kl}(s,\psi)\, \delta^4\left(p_k^{'} + p_l^{'} - p_k - p_l\right)
\end{aligned}$$

with the invariant cross section

$$\sigma^{kl}(s,\psi) = \frac{e_k^2 e_l^2 m_k^2 m_l^2 c^2}{4\pi^2 \epsilon_0^2 \, s} \left|M_{k\,'l\,'kl}\right|^2$$

where s =$p_t^2$ and $p_t = p_k + p_l$. The quantities $p_k$ and $p_l$ are the pre-collisional momenta whereas $p_k'$ and $p_l'$ denotes the post-collisional momenta. The binary transition matrix elements are denoted by $|M_{k'l'kl}|$.

Integrating over $\overrightarrow{p'_t}$we obtain $\overrightarrow{p'_t}$= $\overrightarrow{p_t} - \overrightarrow{p'_k}$. Since the final momentum $p_l'$ has to be on the mass shell we find $(p_t - p_k')^2 = m_l^2 c^2$from which we obtain s + $(m_k^2 - m_l^2)c^2 = 2\, p_k' . p_t$ The quantities $m_k$ and $m_l$ are the pre-collision rest masses of the colliding particles. We find

$$\delta\left(p_k^{0\,'} + p_l^{0\,'} - p_k^0 - p_l^0\right) = \frac{p_k^{0\,'} p_l^{0\,'}}{p_t^0 |\vec{p}_k^{\,'}| - |\vec{p}_t| p_k^{0\,'} \cos\psi} \, \delta\left(|\vec{p}_k^{\,'}| - \mathcal{F}_{kl}\right)$$ where

$$\mathcal{F}_{kl} = \frac{\mathcal{I}\,|\vec{p}_t|\cos\psi}{2\left(s + \vec{p}_t^{\,2}\sin^2\psi\right)} + \sqrt{\left(\frac{\mathcal{I}\,|\vec{p}_t|\cos\psi}{2\left(s + \vec{p}_t^{\,2}\sin^2\psi\right)}\right)^2 + \frac{\mathcal{I}^2 - 4\, m_k^2 c^2\, p_t^{02}}{4\left(s + \vec{p}_t^{\,2}\sin^2\psi\right)}}$$

with I = s + $(m_k^2 - m_l^2)\, c^2$. The angle / denotes the angle between $\overrightarrow{p_t}$ and $\overrightarrow{p'_k}$ . Making use of Equation (3.14) we obtain

$$C_{kl} = \frac{e_k^2 e_l^2 m_k^2 m_l^2 c^2}{4\pi^2 \epsilon_0^2} \int \frac{d^3 p_l}{p_l^0} \int d\Omega_\psi \frac{|\vec{p}_k^{\,\prime}|^2}{p_t^0 |\vec{p}_k^{\,\prime}| - |\vec{p}_t| p_k^{0\,\prime} \cos\psi} \times |M_{k'l'kl}|^2 \left( f_k^{\prime} f_l^{\prime} - f_k f_l \right) ,$$

where $|\overrightarrow{p'_k}| = F_{kl}$ holds. The quantity $d\Omega_\Psi$ denotes an element of solid angle between $\overrightarrow{p_t}$ and $\overrightarrow{p'_k}$. As an example the transition matrix elements for elastic binary collisions are given

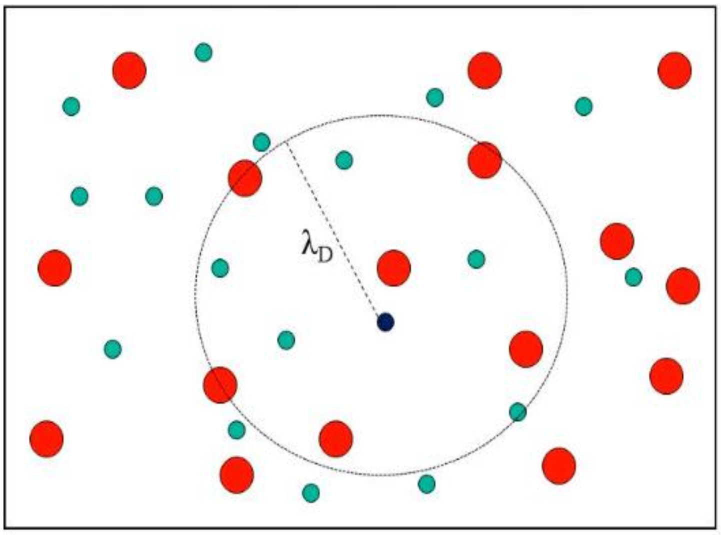
$$\begin{aligned} |M_{k'l'kl}|^2 &= \frac{1}{4} \sum_{s_k' s_l' s_k s_l} \left| \bar{u}_k^{\prime} \gamma_\mu u_k \frac{1}{t + i\epsilon} \bar{u}_l^{\prime} \gamma^\mu u_l \right|^2 \\ &= \frac{(p_l^{\prime} \cdot p_k^{\prime})(p_l \cdot p_k) + (p_l^{\prime} \cdot p_k)(p_l \cdot p_k^{\prime})}{2 m_k^2 m_l^2 c^4 t^2} \\ &\quad - \frac{m_k^2 c^2 (p_l^{\prime} \cdot p_l) + m_l^2 c^2 (p_k^{\prime} \cdot p_k) - 2 m_k^2 m_l^2 c^4}{2 m_k^2 m_l^2 c^4 t^2} , \end{aligned}$$

where $t = (p_k - p_k')^2$ holds. The pre- and post-collision masses are the same. To perform the angle integration we introduce a coordinate system whose polar axis is parallel to $\overrightarrow{p_t}$.

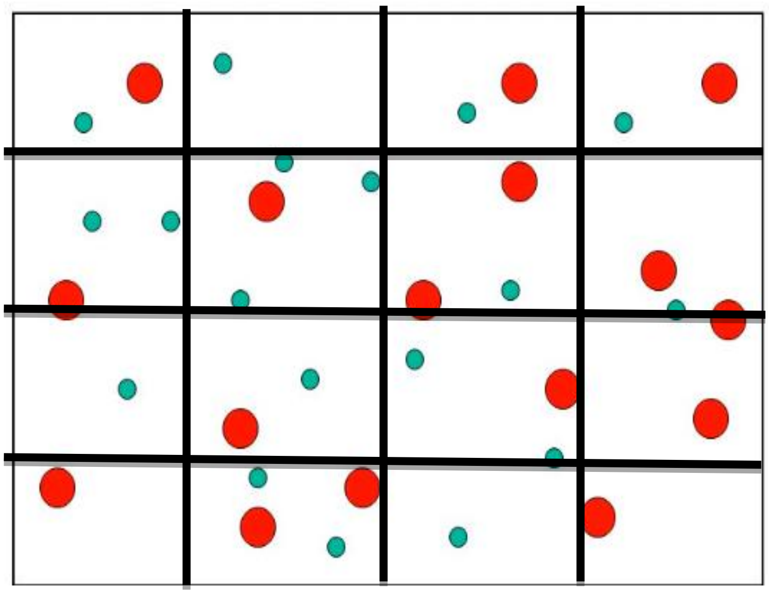
# 3 Numerical model

## 3.1 Meshing

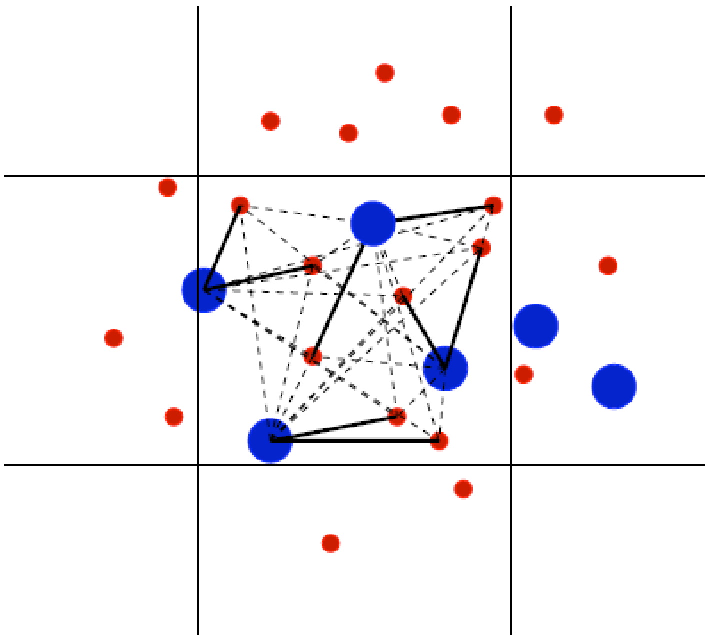
Each particle in the plasma has the probability to collide with other particles in a considered volume.



The grid considered in this program is cubic.

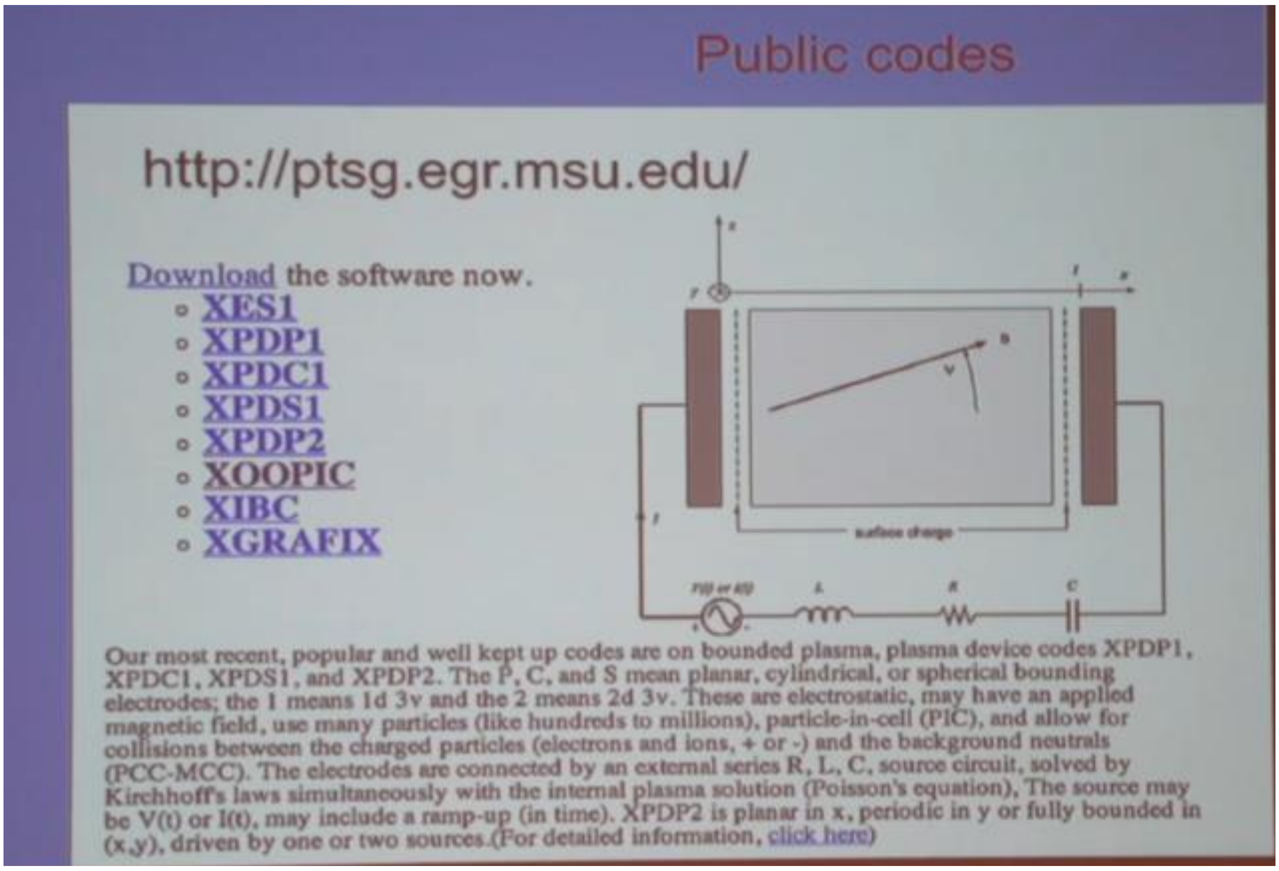


The equations are applied to each cell.



## 4 Some public codes / research groups

http://ptsg.egr.msu.edu



Public codes

http://ptsg.egr.msu.edu/

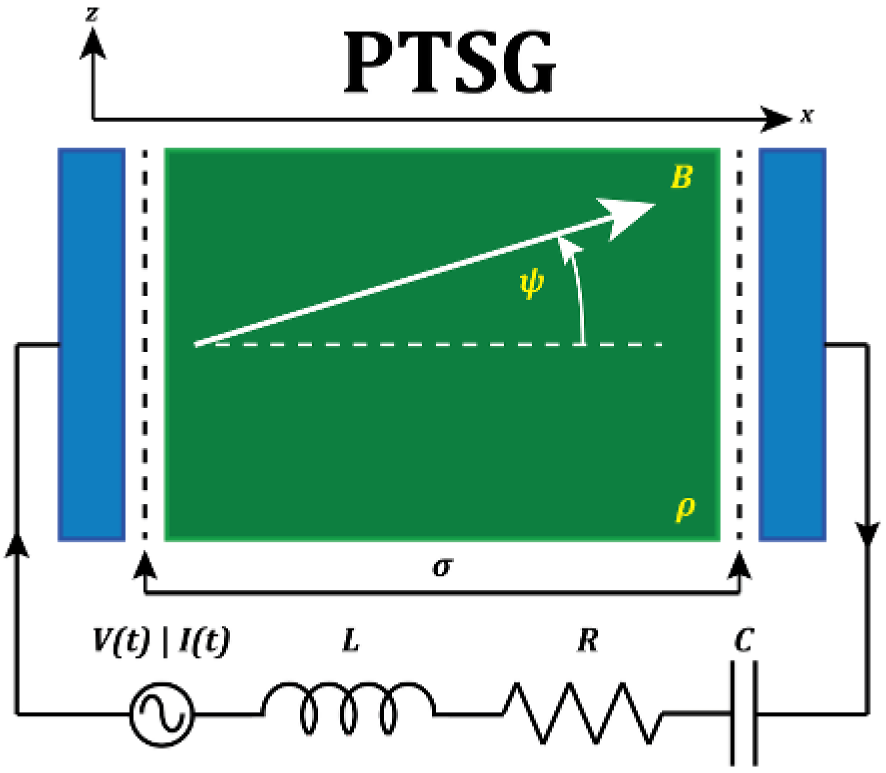
Download the software now.

- XES1
- XPDP1
- XPDC1
- XPDS1
- XPDP2
- XOOPIC
- XIBC
- XGRAFIX

surface charge
L
R
C

Our most recent, popular and well kept up codes are on bounded plasma, plasma device codes XPDP1, XPDC1, XPDS1, and XPDP2. The P, C, and S mean planar, cylindrical, or spherical bounding electrodes; the 1 means 1d 3v and the 2 means 2d 3v. These are electrostatic, may have an applied magnetic field, use many particles (like hundreds to millions), particle-in-cell (PIC), and allow for collisions between the charged particles (electrons and ions, + or -) and the background neutrals (PCC-MCC). The electrodes are connected by an external series R, L, C, source circuit, solved by Kirchhoff's laws simultaneously with the internal plasma solution (Poisson's equation), The source may be V(t) or I(t), may include a ramp-up (in time). XPDP2 is planar in x, periodic in y or fully bounded in (x,y), driven by one or two sources.(For detailed information, click here)

### 4.1 The Plasma Theory and Simulation Group PTSG Univ. of California, Berkley



**PEOPLE PROJECTS SOFTWARE PUBLICATIONS COURSES WORKSHOPS DOWNLOAD COMMENT**

### 4.1.1 People of PTSG

1 Professor, 2 Postdoctoral Researchers, 2 Graduate Students

### 4.1.2 Projects

#### 4.1.2.1 Current Projects in PTSG

**AFOSR:** Consortium on Cathodes and Breakdown for High Power Microwave Devices

**AFOSR/Calabazas Creek Research:** High Voltage Computar Laboratory

**DOE/LLNL:** Hydrocarbon Transport in the Diverter Sheath

**DOE/Calabazas Creek Research:** A Finite Element PIC Model

#### 4.1.2.2 Recently Completed Projects in PTSG

**DOE:** Modeling and Simulation of Plasma Edge Behavior: Formation, Stability and Heating

**ONR:** Plasma Boundaries, Neutral & Non-neutral Plasma, Plasma Devices

**ONR/NRL:** Simulation of Low Frequency Noise in a Coupled Cavity Traveling Wave Tube

**TECH-X:** Object-Oriented PIC Code With Upgraded Physics And Platform Independent GUI

**Hitachi (Japan):** Modeling and Simulation of Plasma Display Panels (PDP's)

**Sumitomo Metals (Japan):**Elecromagnetic Modeling and Simulation of Sumitomo Surface Wave Plasma (SWP) Device

**LLNL (Livermore):**Plasma bulk (fluids), plasma sheath (particles); bounding wall (profiler); an attempt to construct seamless boundaries at the fluid /particle/ wall boundaries, and to run at fluid code speeds

### 4.1.3 PTSG Software

- **General Information**

Our practice has been to make all software developed by PTSG freely available to anyone. If you use our codes or our graphics (both are copyrighted), then please acknowledge PTSG in your publications and send us a copy of your journal articles or reports (send to Prof. John P. Verboncoeur). We would also appreciate receiving copies of your input files (representative) and a copy of code changes which you have made.

- **Acknowledgments**

For our plasma device codes, **XPDP1, XPDP2, and XPDS1**, please acknowledge:

Verboncoeur, J.P., M.V. Alves, V. Vahedi, and C.K. Birdsall, "Simultaneous Potential and Circuit Solution for 1d bounded Plasma Particle Simulation Codes," J. Comp. Physics, 104, pp. 321-328, February 1993.

For **XPDP2**, please acknowledge:

Vahedi, V., C.K. Birdsall, M.A. Lieberman, G. DiPeso, and T.D. Rognlien, "Verification of frequency scaling laws for capacitive radio-frequency discharges using two-dimensional simulations," Phys. Fluids B 5 (7), pp. 2719-2729, July 1993.

V. Vahedi and G. DiPeso, "Simulataneous Potential and Circuit Solution for Two-Dimensional Bounded Plasma Simulation Codes," J. Comp. Phys. 131, pp. 149-163, 1997. (Work begun at U.C. Berkeley)

For **XOOPIC** please acknowledge:

J.P. Verboncoeur, A.B. Langdon and N.T. Gladd, "An Object-Oriented Electromagnetic PIC Code", Comp. Phys. Comm., 87, May11, 1995, pp. 199-211.

- **Description**

Our most recent, popular and well kept up codes are on bounded plasma, plasma device codes XPDP1, XPDC1, XPDS1, and XPDP2. The P, C, and S mean planar, cylindrical, or spherical bounding electrodes; the 1 means 1d 3v and the 2 means 2d 3v. These are electrostatic, may have an applied magnetic field, use many particles (like hundreds to millions), particle-in-cell (PIC), and allow for collisions between the charged particles (electrons and ions, + or -) and the background neutrals (PCC-MCC). The electrodes are connected by an external series R, L, C, source circuit, solved by Kirchhoff's laws simultaneously with the internal plasma solution (Poisson's equation), The source may be V(t) or I(t), may include a ramp-up (in time). XPDP2 is planar in x, periodic in y or fully bounded in (x,y), driven by one or two sources (for detailed information, click here).

Our older, far less well kept up codes are:

**XES1**

**XIBC**

- **Distribution**

All PTSG software is distributed as source codes in form of tarball file. To run the code, one have to have an access to computer with development environment (C/C++ compilers, header files) to be able to compile the code in their system. Source code are available for free download from our website (ptsg.egr.msu.edu). If you would require older version of one of the codes, please contact PTSG Support for further info on how to acquire it.

| Codes that require xgrafix | Codes with own version of xgrafix | Python/Matlab based codes |
|---|---|---|
| Following codes require system-wide installed **xgrafix** (usually in /usr/local/lib). Before compiling codes, please download, compile and install **xgrafix**.<br>xoopic<br>oopd1<br>xpdc1<br>xpdc2<br>xpdp1<br>xpdp2<br>xpds1 | Following codes do not work with current version of xgrafix and have version of xgrafix packed into tarball.<br>xem1<br>xes1<br>xibc | Following codes require working **Python**environment (python 2.7/iython) or MatLab.<br>**pypd1**: Python based code, based on oopd1 (library version of oopd1 is already included in distribution)<br>**s parmos**: MatLab code (requires MatLab) |

### Code Consulting and Maintanence

PTSG may answer short inquiries on our various codes, as we are primarily committed to university type plasma device research and development. However, more detailed inquiries for help should be addressed to Research Institute for Science and Engineering, a small private firm well skilled in PTSG codes: rise@ieee.org.

If you find bugs while using our program, please report them to us. We appreciate these information. Click here for the bug report form.

### 4.1.4 Publications Using PTSG Codes, Worldwide

Many authors, using our codes, have made acknowledgments to PTSG and sent us copies of their reports, journal articles, letters, and MS, Ph.D. theses. We appreciate both.

Since 1991, we have listed the publications brought to our attention using our codes XPDP1, XPDC1, XPDS1, XPDP2, and XOOPIC (by us and others). The total that we are aware of now exceeds 300, including over 60 by us. We are delighted to have been helpful to these authors. We are even more delighted to learn of their excellent and ingenious applications and modifications of our codes (go to the List of the publications).

#### 4.1.5 Workshops

##### 4.1.5.1 Plasma Device Workshop 2009 (PDW2009)

We have given workshops for users of our plasma device codes. These codes are particle-in-cell (PIC) codes with Monte Carlo collision (MCC) models in 1D and 2D. Cartesian, cylindrical, and spherical coordinates are modeled, in both electrostatic and electromagnetic regimes. The models include flexible boundary conditions, external driving circuits, and user-customizable diagnostics. The PTSG has been at the forefront of algorithm development for almost 50 years, and continues to develop new models for physics, computer science, and applied mathematics.

Inquiries are welcome (address to Prof. John P. Verboncoeur).

## 5 Comparison between actual IAP-PCS and XPDP2

| | | |
|---|---|---|
| xgrafix.tar.gz | xpdp2.tar.gz | IAP-PSC_220819_src+exe+output.zip |

### 5.1 Description of XPDP2

For **XPDP2**, please acknowledge:

[1] Vahedi, V., C.K. Birdsall, M.A. Lieberman, G. DiPeso, and T.D. Rognlien, "Verification of frequency scaling laws for capacitive radio-frequency discharges using two-dimensional simulations," Phys. Fluids B 5 (7), pp. 2719-2729, July 1993.

[2] V. Vahedi and G. DiPeso, "Simulataneous Potential and Circuit Solution for Two-Dimensional Bounded Plasma Simulation Codes," J. Comp. Phys. 131, pp. 149-163, 1997. (Work begun at U.C. Berkeley)

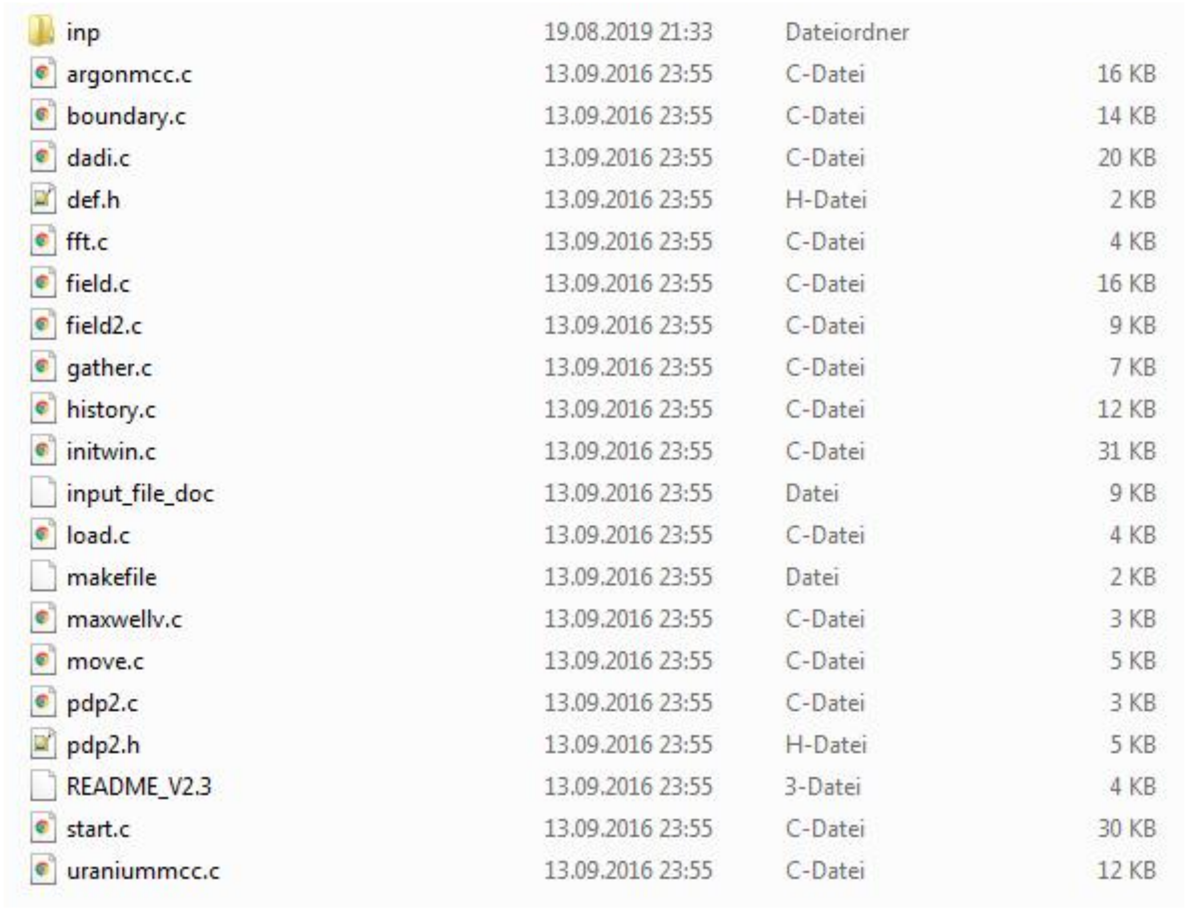
#### 5.1.1 Abstract of [1]:

Weakly ionized processing plasmas are studied in two dimensions using a bounded particle-in-cell (PIC) simulation code with a Monte Carlo collision (MCC) package. The MCC package models the collisions between charged and neutral particles, which are needed to obtain a self-sustained plasma and the proper electron and ion energy loss mechanisms. A two-dimensional capacitive radio-frequency (rf) discharge is investigated in detail. Simple frequency scaling laws for predicting the behavior of some plasma parameters are derived and then compared with simulation results, finding good agreements. It is found that as the drive frequency increases, the sheath width decreases, and the bulk plasma becomes more uniform, leading to a reduction of the ion angular spread at the target and an improvement of ion dose uniformity at the driven electrode.

#### 5.1.2 Abstract of [2]:

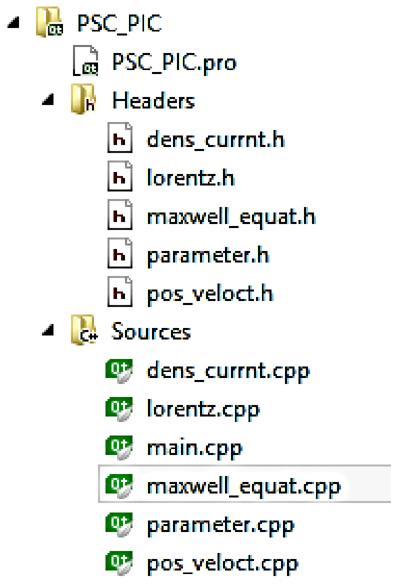
An algorithm for coupling external circuit elements to be bounded two-dimensional electrostatic plasma simulation codes is developed. In general, the external circuit equations provide a mixture of Dirichlet and Neumann boundary conditions for the Poisson equation, which is solved each time step for the internal plasma potential. We rewrite the coupling between the plasma and the external circuit parameters as an algebraic or ordinary differential equation for the potential on the boundary. This scheme allows decomposition of the field solve into a Laplace solver with boundary conditions (e.g., applied potentials) and a Poisson solver with zero boundary conditions. We present the details of the external circuit coupling to an explicit electrostatic planar two-dimensional particle-in-cell code called PDP2, and discuss briefly how the coupling can be done in an implicit electrostatic code. The decomposition replaces the iterative coupling with a direct coupling and reduces the amount of computational time spent in the field solver. We use PDP2 to simulate a dually excited capacitively coupled RF discharge and show how such a system can be used as a plasma processing tool with separate control over ion flux and ion bombarding energy.

### 5.1.3 xpdp2 Code Description



| | | | |
|---|---|---|---|
| inp | 19.08.2019 21:33 | Dateiordner | |
| argonmcc.c | 13.09.2016 23:55 | C-Datei | 16 KB |
| boundary.c | 13.09.2016 23:55 | C-Datei | 14 KB |
| dadi.c | 13.09.2016 23:55 | C-Datei | 20 KB |
| def.h | 13.09.2016 23:55 | H-Datei | 2 KB |
| fft.c | 13.09.2016 23:55 | C-Datei | 4 KB |
| field.c | 13.09.2016 23:55 | C-Datei | 16 KB |
| field2.c | 13.09.2016 23:55 | C-Datei | 9 KB |
| gather.c | 13.09.2016 23:55 | C-Datei | 7 KB |
| history.c | 13.09.2016 23:55 | C-Datei | 12 KB |
| initwin.c | 13.09.2016 23:55 | C-Datei | 31 KB |
| input_file_doc | 13.09.2016 23:55 | Datei | 9 KB |
| load.c | 13.09.2016 23:55 | C-Datei | 4 KB |
| makefile | 13.09.2016 23:55 | Datei | 2 KB |
| maxwellv.c | 13.09.2016 23:55 | C-Datei | 3 KB |
| move.c | 13.09.2016 23:55 | C-Datei | 5 KB |
| pdp2.c | 13.09.2016 23:55 | C-Datei | 3 KB |
| pdp2.h | 13.09.2016 23:55 | H-Datei | 5 KB |
| README_V2.3 | 13.09.2016 23:55 | 3-Datei | 4 KB |
| start.c | 13.09.2016 23:55 | C-Datei | 30 KB |
| uraniummcc.c | 13.09.2016 23:55 | C-Datei | 12 KB |

Our program is composed from:

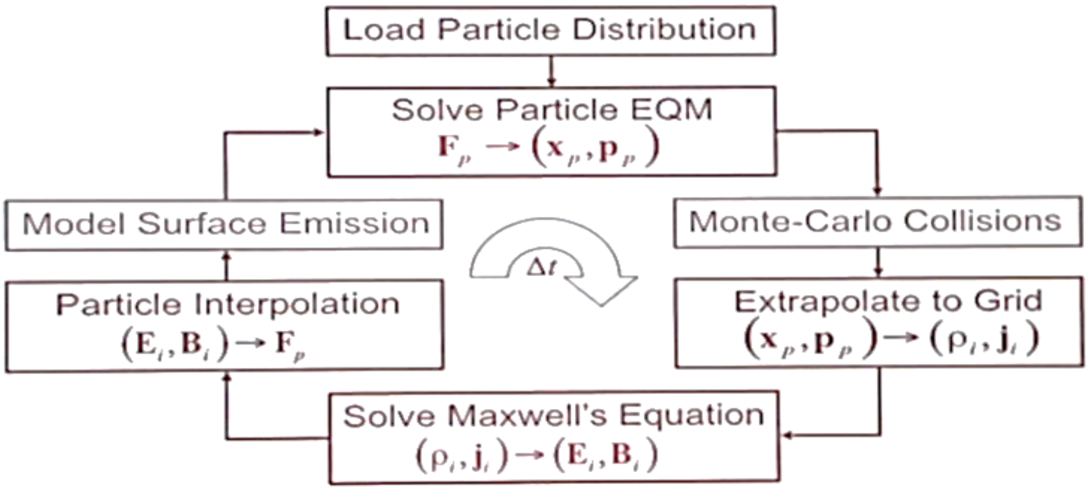


```
PSC_PIC
    PSC_PIC.pro
    Headers
        dens_currnt.h
        lorentz.h
        maxwell_equat.h
        parameter.h
        pos_veloct.h
    Sources
        dens_currnt.cpp
        lorentz.cpp
        main.cpp
        maxwell_equat.cpp
        parameter.cpp
        pos_veloct.cpp
```

The mesh is a cube divided into 5 parts on each side.

It contains 1000 particles, 500 electrons and 500 ions.

The sequence followed is:


Load Particle Distribution
Solve Particle EQM
$\mathbf{F}_p \rightarrow (\mathbf{x}_p, \mathbf{p}_p)$
Monte-Carlo Collisions
Extrapolate to Grid
$(\mathbf{x}_p, \mathbf{p}_p) \rightarrow (\rho_i, \mathbf{j}_i)$
Solve Maxwell's Equation
$(\rho_i, \mathbf{j}_i) \rightarrow (\mathbf{E}_i, \mathbf{B}_i)$
Particle Interpolation
$(\mathbf{E}_i, \mathbf{B}_i) \rightarrow \mathbf{F}_p$
Model Surface Emission
$\Delta t$

## These equations are applied:

- Lorentz-Force: $\mathbf{F}_p = q\mathbf{E}_p + \frac{q}{m}(\mathbf{p}_p \times \mathbf{B}_p)$

## Position and velocity:

$$\frac{d\vec{v}_j}{dt} = \frac{q_j}{m_j}(\vec{E} + \frac{\vec{v}_j \times \vec{B}}{c})$$

$$\frac{d\vec{x}}{dt} = \vec{v}$$

## Density and current:

$$\rho(\vec{x}) = \sum_j q_j \delta(\vec{x} - \vec{x}_j)$$

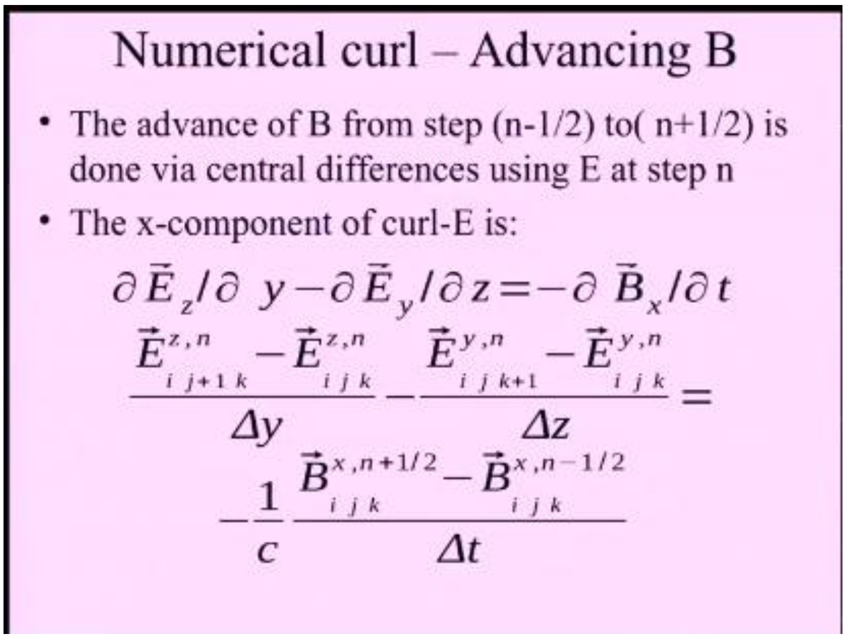
$$\vec{j}(\vec{x}) = \sum_j q_j \vec{v}_j \delta(\vec{x} - \vec{x}_j)$$

## Maxwell equations to calculate E and B

$$\nabla \times \vec{E} = -\frac{1}{c}\frac{\partial \vec{B}}{\partial t}$$

$$\nabla \cdot \vec{B} = 0, \quad \nabla \cdot \vec{E} = 4\pi\rho$$

$$\nabla \times \vec{B} = \frac{4\pi\vec{j}}{c} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t}$$ [1]



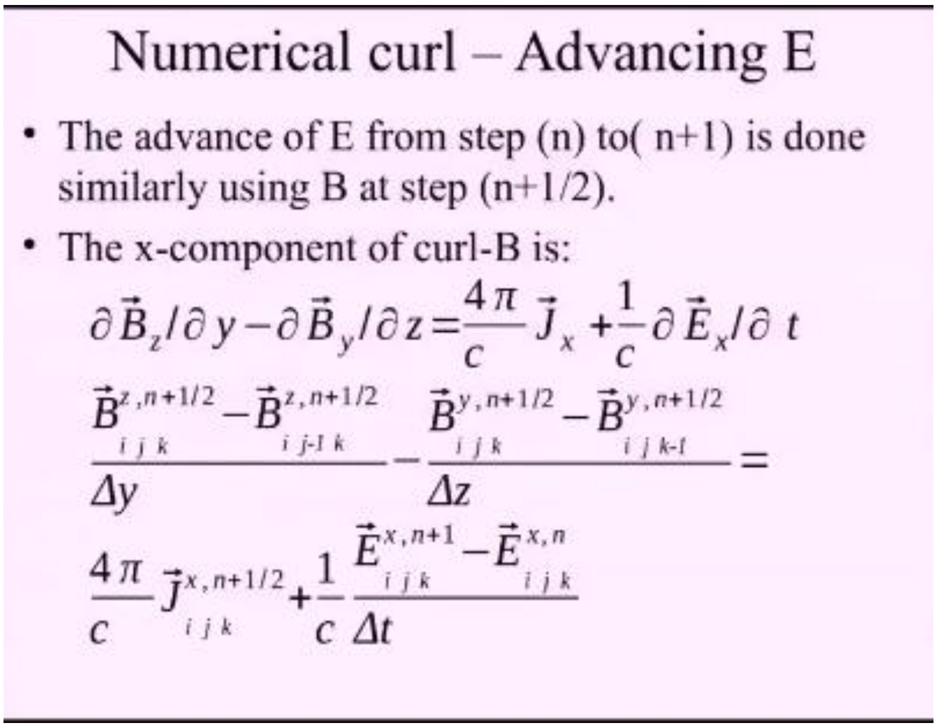
## Numerical curl – Advancing B

- The advance of B from step (n-1/2) to( n+1/2) is done via central differences using E at step n
- The x-component of curl-E is:

$$\partial \vec{E}_z/\partial\ y - \partial \vec{E}_y/\partial z = -\partial\ \vec{B}_x/\partial t$$

$$\frac{\vec{E}^{z,n}_{i\ j+1\ k} - \vec{E}^{z,n}_{i\ j\ k}}{\Delta y} - \frac{\vec{E}^{y,n}_{i\ j\ k+1} - \vec{E}^{y,n}_{i\ j\ k}}{\Delta z} =$$

$$-\frac{1}{c}\,\frac{\vec{B}^{x,n+1/2}_{i\ j\ k} - \vec{B}^{x,n-1/2}_{i\ j\ k}}{\Delta t}$$



## Numerical curl – Advancing E

- The advance of E from step (n) to( n+1) is done similarly using B at step (n+1/2).
- The x-component of curl-B is:

$$\partial \vec{B}_z/\partial y - \partial \vec{B}_y/\partial z = \frac{4\pi}{c}\vec{J}_x + \frac{1}{c}\partial \vec{E}_x/\partial\ t$$

$$\frac{\vec{B}^{z,n+1/2}_{i\ j\ k} - \vec{B}^{z,n+1/2}_{i\ j\text{-}1\ k}}{\Delta y} - \frac{\vec{B}^{y,n+1/2}_{i\ j\ k} - \vec{B}^{y,n+1/2}_{i\ j\ k\text{-}1}}{\Delta z} =$$

$$\frac{4\pi}{c}\vec{J}^{x,n+1/2}_{i\ j\ k} + \frac{1}{c}\,\frac{\vec{E}^{x,n+1}_{i\ j\ k} - \vec{E}^{x,n}_{i\ j\ k}}{\Delta t}$$

[1] https://www.youtube.com/watch?v=I09QeVDoEZY

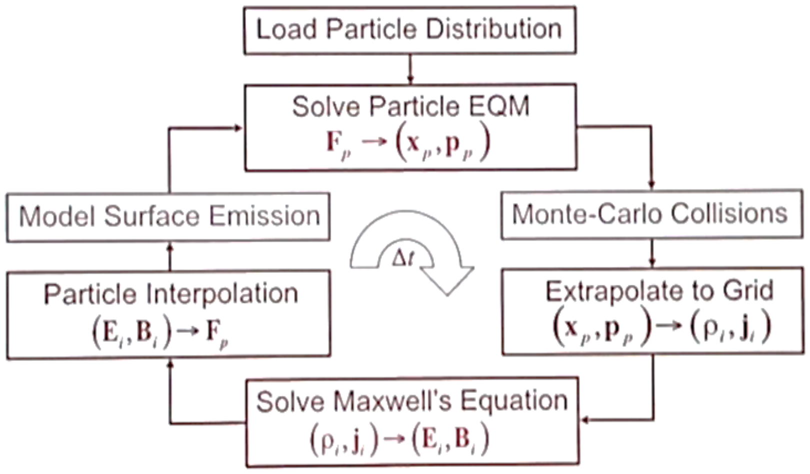
| XPDP2 file | Description | | IAP-PSC file | Description |
|---|---|---|---|---|
| inp/argon.inp | argon.inp: Argon RF discharge<br><br>-nsp---ncx---ncy---nc2p---dt[s]---xlength[m]--ylength[m]--zlength[m]--epsilonr-<br>2 80 160 1e6 3.600894e-11 0.03 0.06 0.1 1.0<br>ELECTRONS (Species 1)<br><br>----q[C]-------m[Kg]-------k-----max-np---rel_weight--<br>-1.602e-19 9.11e-31 1 200000 1 | | Main.cpp | Call all the functions , advance their values to the next time step |
| inp/maxwell.inp | | | | |
| maxwellv.c | `void maxwellv(SCALAR *vx, SCALAR *vy, SCALAR *vz, SCALAR vth, SCALAR Rv, SCALAR Rphi, SCALAR Rthe)` | | maxwell_equat.h<br><br>maxwell_equat.cpp | $\nabla \times \vec{E} = -\frac{1}{c}\frac{\partial \vec{B}}{\partial t}$<br>$\nabla \cdot \vec{B} = 0, \quad \nabla \cdot \vec{E} = 4\pi\rho$<br>$\nabla \times \vec{B} = \frac{4\pi\vec{j}}{c} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t}$ |
| Move.c | | | Pos_veloct.h<br><br>Pos_veloct.c | $\frac{d\vec{v}_j}{dt} = \frac{q_j}{m_j}(\vec{E} + \frac{\vec{v}_j \times \vec{B}}{c})$<br>$\frac{d\vec{x}}{dt} = \vec{v}$ |
| | | | Lorentz.h, Lorentz.c | $\mathbf{F}_p = q\mathbf{E}_p + \frac{q}{m}(\mathbf{p}_p \times \mathbf{B}_p)$ |
| | | | Parameter.h, Parameter.c | Initial and boundary conditions |
| | | | Dens_currnt.h<br><br>Dens_currnt.c | $\rho(\vec{x}) = \sum_j q_j \delta(\vec{x} - \vec{x}_j)$<br>$\vec{j}(\vec{x}) = \sum_j q_j \vec{v}_j \delta(\vec{x} - \vec{x}_j)$ / |

# 6 IAP-PSC Program

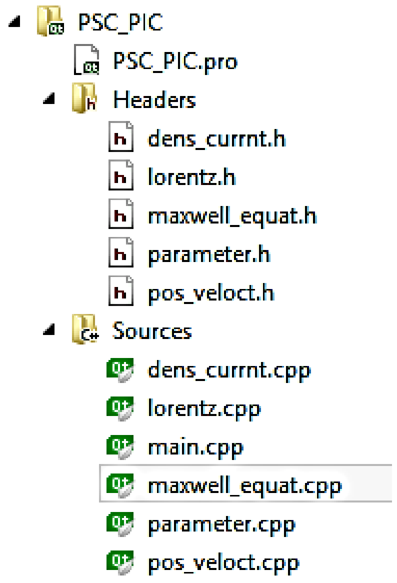
## 6.1 Description

Our program is written with c++ on Qt creator and the results were presented on Paraview.

The program study the comportment of a plasma composed of 1000 particles (500 electrons and 500 ions) placed randomly in a box under the reaction of electric and magnetic field. The program follows the chart below:


Load Particle Distribution
Solve Particle EQM
$F_p \rightarrow (x_p, p_p)$
Model Surface Emission
Monte-Carlo Collisions
$\Delta t$
Particle Interpolation
$(E_i, B_i) \rightarrow F_p$
Extrapolate to Grid
$(x_p, p_p) \rightarrow (\rho_i, j_i)$
Solve Maxwell's Equation
$(\rho_i, j_i) \rightarrow (E_i, B_i)$

It is composed from 5 classes and the main program.



```
PSC_PIC
    PSC_PIC.pro
    Headers
        dens_currnt.h
        lorentz.h
        maxwell_equat.h
        parameter.h
        pos_veloct.h
    Sources
        dens_currnt.cpp
        lorentz.cpp
        main.cpp
        maxwell_equat.cpp
        parameter.cpp
        pos_veloct.cpp
```

After initializing the parameter and the initial condition using the class << **parameter.cpp** >>, the Lorentz force on each particle is calculated using the class **<< Lorentz.cpp>>** then the new position and the velocity is calculated using the class << **pos_veloct.cpp** >>. The density and the current on the grid are calculated using the class << **dens_currnt.**cpp>>. Using these values and the Maxwell equations, the electric and magnetic field are calculated on each cell using the class << **Maxwell_equat.cpp** >>. Now the Lorentz force is recalculated and a new time step starts.

Our program repeats this chain for 10 time steps.

## 6.2 The Code

### 6.2.1 Class 1: parameters

**Parameter.h**

```
#ifndef PARAMETER_H
#define PARAMETER_H


class parameter
{
public:
    parameter(){

    }

    void init_elect_param(double xe[500][10],double ye[500][10],double
ze[500][10],double vex[500][10],double vey[500][10],double vez[500][10],double
Fex[500][10],double Fey[500][10],double Fez[500][10],double Eex[500][10],double
Eey[500][10],double Eez[500][10],double Bex[500][10],double Bey[500][10],double
Bez[500][10]);
    void init_ion_param(double xi[500][10],double yi[500][10],double
zi[500][10],double vix[500][10],double viy[500][10],double viz[500][10],double
Fix[500][10],double Fiy[500][10],double Fiz[500][10],double Eix[500][10],double
Eiy[500][10],double Eiz[500][10],double Bix[500][10],double Biy[500][10],double
Biz[500][10]);
    void init_EB(double Ex[5][5][5][10],double Ey[5][5][5][10],double
Ez[5][5][5][10],double Bx[5][5][5][10],double By[5][5][5][10],double
Bz[5][5][5][10],double rhox[5][5][5][10],double rhoy[5][5][5][10],double
rhoz[5][5][5][10],double Jx[5][5][5][10],double Jy[5][5][5][10],double
Jz[5][5][5][10]);

};

#endif // PARAMETER_H
```

**Parameter.cpp**

```
#include "parameter.h"
#include <cstdlib>
void parameter::init_elect_param(double xe[500][10],double ye[500][10],double
ze[500][10],double vex[500][10],double vey[500][10],double vez[500][10],double
Fex[500][10],double Fey[500][10],double Fez[500][10],double Eex[500][10],double
Eey[500][10],double Eez[500][10],double Bex[500][10],double Bey[500][10],double
Bez[500][10]){

int a=0, b=0, c=0;
      for(int l=0;l<500;l++){
          a=rand()%1000;
          b=rand()%1000;
          c=rand()%1000;

          xe[l][0]=a;

               ye[l][0]=b;

                  ze[l][0]=c;


          }
```

```
 for(int i=0;i<500;i++){

        vex[i][0]=2.42;//m/s
        vey[i][0]=2.42;
        vez[i][0]=2.42;
        Fex[i][0]=0;
        Fey[i][0]=0;
        Fez[i][0]=0;
        Eex[i][0]=2000;
        Eey[i][0]=2000;
        Eez[i][0]=2000;
        Bex[i][0]=47*10e-6;
        Bey[i][0]=47*10e-6;
        Bez[i][0]=47*10e-6;

    }
}
void parameter::init_ion_param(double xi[500][10],double yi[500][10],double
zi[500][10],double vix[500][10],double viy[500][10],double viz[500][10],double
Fix[500][10],double Fiy[500][10],double Fiz[500][10],double Eix[500][10],double
Eiy[500][10],double Eiz[500][10],double Bix[500][10],double Biy[500][10],double
Biz[500][10]){
    int a=0, b=0, c=0;
          for(int l=0;l<500;l++){
              a=rand()%1000;
              b=rand()%1000;
              c=rand()%1000;

              xi[l][0]=a;
                   yi[l][0]=b;
                      zi[l][0]=c;


              }

        for(int i=0;i<500;i++){
        vix[i][0]=1;
        viy[i][0]=1;
        viz[i][0]=1;
        Fix[i][0]=0;
        Fiy[i][0]=0;
        Fiz[i][0]=0;
        Eix[i][0]=2000;
        Eiy[i][0]=2000;
        Eiz[i][0]=2000;
        Bix[i][0]=47*10e-6;
        Biy[i][0]=47*10e-6;
        Biz[i][0]=47*10e-6;



    }
}
void parameter::init_EB(double Ex[5][5][5][10],double Ey[5][5][5][10],double
Ez[5][5][5][10],double Bx[5][5][5][10],double By[5][5][5][10],double
Bz[5][5][5][10],double rhox[5][5][5][10],double rhoy[5][5][5][10],double
rhoz[5][5][5][10],double Jx[5][5][5][10],double Jy[5][5][5][10],double
Jz[5][5][5][10]){
    for(int i=0;i<5;i++){
          for(int j=0;j<5;j++){
                for(int k=0;k<5;k++){
          Ex[i][0][0][0]=10e5;
          Ey[0][j][0][0]=10e5;
          Ez[0][0][k][0]=10e5;
          Bx[0][j][k][0]=1*10e-6;
          By[i][0][k][0]=1*10e-6;
          Bz[i][j][0][0]=1*10e-6;
          rhox[i][j][k][0]=0;
          rhoy[i][j][k][0]=0;
          rhoz[i][j][k][0]=0;
          Jx[i][j][k][0]=0;
          Jy[i][j][k][0]=0;
          Jz[i][j][k][0]=0;
   }
          }
    }

}
```

### 6.2.2 Class 2: Lorentz

**Lorentz.h**

```
#ifndef LORENTZ_H
#define LORENTZ_H


class lorentz
{
public:
    lorentz(){

    }

    double Force (double q,double Ep,double m,double p,double Bp);
};

#endif // LORENTZ_H
```

**Lorentz.cpp**

```
#include "lorentz.h"
#include "math.h"


 double lorentz::Force (double q,double Ep,double m,double p,double Bp)
{
double F;
F= q*Ep+q/m*(p*Bp);

return F;
}
```

### 6.2.3 Class 3: pos_veloct

**Pos_veloct.h**

```
#ifndef POS_VELOCT_H
#define POS_VELOCT_H


class Pos_veloct
{
public:
    Pos_veloct(){
    }


    double velocity(double F,double m);

};

#endif // POS_VELOCT_H
```

**Pos_veloct.cpp**

```
#include "pos_veloct.h"

double Pos_veloct::velocity(double F,double m){
   double dvdt;
    dvdt=F/m;
    return dvdt;
}
```

### 6.2.4 Class 4: Dens_current

**Dens_current.h:**

```
#ifndef DENS_CURRNT_H
#define DENS_CURRNT_H


class dens_currnt
{
public:
    dens_currnt(){

    }

    double dens(double q,double i,double x,double delta);
    double curr(double q,double i,double x,double delta, double v);
};
```

#endif // DENS_CURRNT_H

**Dens_currnt.cpp**

```
#include "dens_currnt.h"

double dens_currnt::dens(double q,double i,double x,double delta){
    double dens;
    dens=q*delta*(i-x);
    return dens;
}
double dens_currnt::curr(double q,double i,double x,double delta, double v){
    double curr;
    curr=q*v*delta*(i-x);
    return curr;
}
```

### 6.2.5 Class 5: Maxwell_equat

**Maxwell_equat.h**

```
#ifndef MAXWELL_EQUAT_H
#define MAXWELL_EQUAT_H
```

```
class maxwell_equat
{
public:
    maxwell_equat(){

    }

    double electx(double ex,double c,int delta_y,int delta_z,int delta_t,double
Bz,double bz, double By,double by,double J,double PI );
    double electy(double ey,double c,int delta_x,int delta_z,int delta_t,double
Bz,double bz, double Bx,double bx,double J,double PI );
    double electz(double ez,double c,int delta_y,int delta_x,int delta_t,double
Bx,double bx, double By,double by,double J,double PI );

    double magntx( double bx, double c,double Ez,double ez,double Ey,double
ey,int delta_y,int delta_z,int delta_t);
    double magnty( double by, double c,double Ez,double ez,double Ex,double
ex,int delta_x,int delta_z,int delta_t);
    double magntz( double bz, double c,double Ex,double ex,double Ey,double
ey,int delta_x,int delta_y,int delta_t);

};

#endif // MAXWELL_EQUAT_H
```

**Maxwell_equat.cpp**

```
#include "maxwell_equat.h"
#include "math.h"

double maxwell_equat::electx(double ex,double c,int delta_y,int delta_z,int
delta_t,double Bz,double bz, double By,double by,double J,double PI ){
    double Ex;
    Ex=ex-delta_t*4*PI*J+(delta_t*c/delta_y)*(Bz-bz)-(delta_t*c/delta_z)*(By-
by);
return Ex;
}
    double maxwell_equat::electy(double ey,double c,int delta_x,int delta_z,int
delta_t,double Bz,double bz, double Bx,double bx,double J,double PI ){
double Ey;
    Ey=ey-delta_t*4*PI*J+(delta_t*c/delta_x)*(Bz-bz)-(delta_t*c/delta_z)*(Bx-
bx);
    return Ey;
    }
    double maxwell_equat::electz(double ez,double c,int delta_y,int delta_x,int
delta_t,double Bx,double bx, double By,double by,double J,double PI ){
double Ez;
    Ez=ez-delta_t*4*PI*J+(delta_t*c/delta_y)*(Bx-bx)-(delta_t*c/delta_x)*(By-
by);
return Ez;
}
double maxwell_equat::magntx( double bx, double c,double Ez,double ez,double
Ey,double ey,int delta_y,int delta_z,int delta_t){
    // x component
    double Bx;
    Bx=bx-(c*delta_t)*((Ez-ez)/delta_y)+(c*delta_t)*((Ey-ey)/(delta_z));
  return Bx;

}
```

```
double maxwell_equat::magnty( double by, double c,double Ez,double ez,double
Ex,double ex,int delta_x,int delta_z,int delta_t){
double By;
// y component
    By=by-(c*delta_t)*((Ez-ez)/delta_x)+(c*delta_t)*((Ex-ex)/(delta_z));
return By;
}
double maxwell_equat::magntz( double bz, double c,double Ex,double ex,double
Ey,double ey,int delta_x,int delta_y,int delta_t){
double Bz;
    // z component
    Bz=bz-(c*delta_t)*((Ex-ex)/delta_y)+(c*delta_t)*((Ey-ey)/(delta_x));
return Bz;
}

//      Bx[i][j][k][t+1]=Bx[i][j][k][t]-(c*delta_t)*((Ez[i][j+1][k][t]-
Ez[i][j][k][t])/delta_y)+(c*delta_t)*((Ey[i][j][k+1][t]-
Ey[i][j][k][t])/(delta_z));
//      By[i][j][k][t+1]=By[i][j][k][t]-(c*delta_t)*((Ez[i+1][j][k][t]-
Ez[i][j][k][t])/delta_x)+(c*delta_t)*((Ex[i][j][k+1][t]-
Ex[i][j][k][t])/(delta_z));
//      Bz[i][j][k][t+1]=Bz[i][j][k][t]-(c*delta_t)*((Ex[i][j+1][k][t]-
Ex[i][j][k][t])/delta_y)+(c*delta_t)*((Ey[i+1][j][k][t]-
Ey[i][j][k][t])/(delta_x));
// Ex[i][j][k][t+1]=Ex[i][j][k][t]-
delta_t*4*PI*Jx[i][j][k][t]+(delta_t*c/delta_y)*(Bz[i][j][k][t+1]-Bz[i][j-
1][k][t+1])-(delta_t*c/delta_z)*(By[i][j][k][t+1]-By[i][j][k-1][t+1]);
// Ey[i][j][k][t+1]=Ey[i][j][k][t]-
delta_t*4*PI*Jy[i][j][k][t]+(delta_t*c/delta_x)*(Bz[i][j][k][t+1]-Bz[i-
1][j][k][t+1])-(delta_t*c/delta_z)*(Bx[i][j][k][t+1]-Bx[i][j][k-1][t+1]);
// Ez[i][j][k][t+1]=Ez[i][j][k][t]-
delta_t*4*PI*Jz[i][j][k][t]+(delta_t*c/delta_y)*(Bx[i][j][k][t+1]-Bx[i][j-
1][k][t+1])-(delta_t*c/delta_x)*(By[i][j][k][t+1]-By[i-1][j][k][t+1]);
```

### 6.2.6 Main program

**Main.cpp**

```
#include <QCoreApplication>
#include<iostream>
#include<fstream>
#include<string>
#include "parameter.h"
#include "lorentz.h"
#include "pos_veloct.h"
#include "dens_currnt.h"
#include "maxwell_equat.h"
#include "math.h"
#include "algorithm"
#include "iostream"
#include "fstream"
#include "sstream"
#include "stdio.h"
#include <QtCore/QString>
#include <QtCore/QFile>
#include <QtCore/QDebug>
#include <QtCore/QTextStream>
#include <cstdlib>
using namespace std;
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    double xe[500][10];
    double ye[500][10];
    double ze[500][10];
    double vex[500][10];
    double vey[500][10];
    double vez[500][10];
    double dvex[500][10];
    double dvey[500][10];
    double dvez[500][10];
    double Fex[500][10];
    double Fey[500][10];
    double Fez[500][10];
    double xi[500][10];
    double yi[500][10];
    double zi[500][10];
    double vix[500][10];
    double viy[500][10];
    double viz[500][10];
    double dvix[500][10];
    double dviy[500][10];
    double dviz[500][10];
    double Fix[500][10];
    double Fiy[500][10];
    double Fiz[500][10];
    double mi=1.67*10e-27,me=9.11*10e-28,qi=1.602*10e-19,qe=-1.602*10e-19;
    double pex[500][10];
    double pey[500][10];
    double pez[500][10];
    double pix[500][10];
    double piy[500][10];
    double piz[500][10];
    double Ex[5][5][5][10];
    double Ey[5][5][5][10];
    double Ez[5][5][5][10];
    double Eix[500][10],Eex[500][10];
    double Eiy[500][10],Eey[500][10];
    double Eiz[500][10],Eez[500][10];
    double Bix[500][10],Bex[500][10];
    double Biy[500][10],Bey[500][10];
    double Biz[500][10],Bez[500][10];
    double Bx[5][5][5][10];
    double By[5][5][5][10];
    double Bz[5][5][5][10];
    double rhox[5][5][5][10];
    double rhoz[5][5][5][10];
    double rhoy[5][5][5][10];
    double Jx[5][5][5][10];
    double Jy[5][5][5][10];
    double Jz[5][5][5][10];
    double delta=0;
    double c=3*10e8,PI=3.14;
    int delta_x=1, delta_y=1, delta_z=1, delta_t=1;

 cout<<"The program has start\n";


// parameter initialization

parameter param;
 param.init_elect_param(xe,ye, ze, vex, vey, vez,
Fex,Fey,Fez,Eex,Eey,Eez,Bex,Bey,Bez);
 param.init_ion_param( xi, yi, zi, vix, viy,viz,
Fix,Fiy,Fiz,Eix,Eiy,Eiz,Bix,Biy,Biz);
```

```
 param.init_EB( Ex, Ey, Ez, Bx, By, Bz,rhox,rhoy,rhoz, Jx,Jy,Jz);

//force de lorentz
for(int t=0;t<10;t++){
 lorentz lort;
 for(int i=0;i<500;i++){
 Fex[i][t]=lort.Force(qe,Eex[i][t],me,pex[i][t],Bex[i][t]);
 Fey[i][t]=lort.Force(qe,Eey[i][t],me,pey[i][t],Bey[i][t]);
 Fez[i][t]=lort.Force(qe,Eez[i][t],me,pez[i][t],Bez[i][t]);
 Fix[i][t]=lort.Force(qi,Eix[i][t],mi,pix[i][t],Bix[i][t]);
 Fiy[i][t]=lort.Force(qi,Eiy[i][t],mi,piy[i][t],Biy[i][t]);
 Fiz[i][t]=lort.Force(qi,Eiz[i][t],mi,piz[i][t],Biz[i][t]);
 }

 //position and velocity

 Pos_veloct vel;
 for (int i=0;i<500;i++){
    dvex[i][t]=vel.velocity(Fex[i][t],me);
    dvey[i][t]=vel.velocity(Fey[i][t],me);
    dvez[i][t]=vel.velocity(Fez[i][t],me);
    dvix[i][t]=vel.velocity(Fix[i][t],mi);
    dviy[i][t]=vel.velocity(Fiy[i][t],mi);
    dviz[i][t]=vel.velocity(Fiz[i][t],mi);

 }
 for(int i=0;i<500;i++){
     vex[i][t+delta_t]= vex[i][t]+((2*delta_t) * dvex[i][t]);
     vey[i][t+delta_t]= vey[i][t]+((2*delta_t) * dvey[i][t]);
     vez[i][t+delta_t]= vez[i][t]+((2*delta_t) * dvez[i][t]);
     vix[i][t+delta_t]= vix[i][t]+((2*delta_t) * dvix[i][t]);
     viy[i][t+delta_t]= viy[i][t]+((2*delta_t) * dviy[i][t]);
     viz[i][t+delta_t]= viz[i][t]+((2*delta_t) * dviz[i][t]);

 }
 for(int i=0;i<500;i++){
      xe[i][t+delta_t]= xe[i][t]+((2*delta_t) * vex[i][t]);
      ye[i][t+delta_t]= ye[i][t]+((2*delta_t) * vey[i][t]);
      ze[i][t+delta_t]= ze[i][t]+((2*delta_t) * vez[i][t]);
      xi[i][t+delta_t]= xi[i][t]+((2*delta_t) * vix[i][t]);
      yi[i][t+delta_t]= yi[i][t]+((2*delta_t) * viy[i][t]);
      zi[i][t+delta_t]= zi[i][t]+((2*delta_t) * viz[i][t]);


 }
 //the particles should not go out the box(boundaries)
 for(int i=0;i<500;i++){
     if(xe[i][t+delta_t]>1000 || xe[i][t+delta_t]<0){
     xe[i][t+delta_t]= rand()%1000;}
     if(ye[i][t+delta_t]>1000 || ye[i][t+delta_t]<0){
     ye[i][t+delta_t]= rand()%1000;}
     if(ze[i][t+delta_t]>1000 || ze[i][t+delta_t]<0){
     ze[i][t+delta_t]= rand()%1000;}
     if(xi[i][t+delta_t]>1000 || xi[i][t+delta_t]<0){
     xi[i][t+delta_t]= rand()%1000;}
     if(yi[i][t+delta_t]>1000 || yi[i][t+delta_t]<0){
     yi[i][t+delta_t]= rand()%1000;}
     if(zi[i][t+delta_t]>1000 || zi[i][t+delta_t]<0 ){
     zi[i][t+delta_t]= rand()%1000;}

 }

// system("pause");
// density and current
dens_currnt dens;
double k1=0,k2=0,k3=0,k4=0,k5=0,k6=0;
double l1=0,l2=0,l3=0,l4=0,l5=0,l6=0;
for(int i=0;i<5;i++){
    for (int j=0;j<5;j++){
     for (int k=0;k<5;k++){
         for (int l=0;l<500;l++) {
         k1+=dens.dens(qe,i,xe[l][t],delta);
         k2+=dens.dens(qe,j,ye[l][t],delta);
         k3+=dens.dens(qe,k,ze[l][t],delta);
         k4+=dens.curr(qe,i,xe[l][t],delta,vex[l][t]);
         k5+=dens.curr(qe,j,ye[l][t],delta,vey[l][t]);
         k6+=dens.curr(qe,k,ze[l][t],delta,vez[l][t]);
     //}
//if(xi[l][t+1]<i+1 && xi[l][t+1]>i && yi[l][t+1]<j+1 && yi[l][t+1]>j &&
zi[l][t+1]<k+1 && zi[l][t+1]>k){
         l1+=dens.dens(qi,i,xi[l][t],delta);
         l2+=dens.dens(qi,j,yi[l][t],delta);
         l3+=dens.dens(qi,k,zi[l][t],delta);
         l4+=dens.curr(qi,i,xi[l][t],delta,vix[l][t]);
         l5+=dens.curr(qi,i,yi[l][t],delta,viy[l][t]);
         l6+=dens.curr(qi,i,zi[l][t],delta,viz[l][t]);
    // }
         }

rhox[i][j][k][t]=(k1+l1);
 rhoy[i][j][k][t]=(k2+l2);
  rhoz[i][j][k][t]=(k3+l3);
Jx[i][j][k][t]=(k4+l4);
 Jy[i][j][k][t]=(k5+l5);
  Jz[i][j][k][t]=(k6+l6);
     }
    }
 }

// electric and magnetic field


maxwell_equat max;

for(int i=0;i<5;i++){
   for (int j=0;j<5;j++){
    for (int k=0;k<5;k++){

Bx[i][j][k][t+1]=max.magntx(Bx[i][j][k][t],c,Ez[i][j+1][k][t],Ez[i][j][k][t],Ey[
i][j][k+1][t],Ey[i][j][k][t],delta_y,delta_z,delta_t);

By[i][j][k][t+1]=max.magnty(By[i][j][k][t],c,Ez[i+1][j][k][t],Ez[i][j][k][t],Ex[
i][j][k+1][t],Ex[i][j][k][t],delta_x,delta_z,delta_t);

Bz[i][j][k][t+1]=max.magntz(Bz[i][j][k][t],c,Ex[i][j+1][k][t],Ex[i][j][k][t],Ey[
i+1][j][k][t],Ey[i][j][k][t],delta_x,delta_y,delta_t);


    }
   }
}

for(int i=0;i<5;i++){
   for (int j=0;j<5;j++){
    for (int k=0;k<5;k++){
```

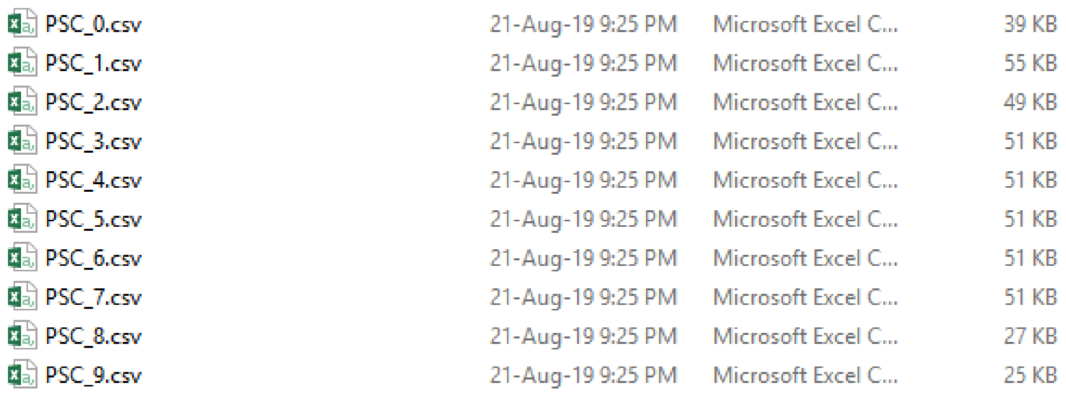
```
Ex[i][j][k][t+1]=max.electx(Ex[i][j][k][t],c,delta_y,delta_z,delta_t,Bz[i][j][k]
[t+1],Bz[i][j-1][k][t+1],By[i][j][k][t+1],By[i][j][k-1][t+1],Jx[i][j][k][t],PI);

Ey[i][j][k][t+1]=max.electy(Ey[i][j][k][t],c,delta_x,delta_z,delta_t,Bz[i][j][k]
[t+1],Bz[i-1][j][k][t+1],Bx[i][j][k][t+1],Bx[i][j][k-1][t+1],Jy[i][j][k][t],PI);

Ez[i][j][k][t+1]=max.electz(Ez[i][j][k][t],c,delta_y,delta_x,delta_t,Bx[i][j][k]
[t+1],Bx[i][j-1][k][t+1],By[i][j][k][t+1],By[i-1][j][k][t+1],Jz[i][j][k][t],PI);

    }
   }
}
double s4=0,s1=0,s2=0,s3=0; //sum
double m1=0,m2=0,m3=0,m4=0;
double n1=0,n2=0,n3=0,n4=0;

//particle interpolation
//x component
for(int l=0;l<500;l++){
    for(int i=0;i<5;i++){
   for (int j=0;j<5;j++){
    for (int k=0;k<5;k++){

s4+=Ex[i][j][k][t+1];
s2+=Bx[i][j][k][t+1];


}
    }
   }
    Eex[l][t+1]=s4;
    Bex[l][t+1]=s2;
}
for(int l=0;l<500;l++){
    for(int i=0;i<5;i++){
   for (int j=0;j<5;j++){
    for (int k=0;k<5;k++){

s1+=Ex[i][j][k][t+1];
s3+=Bx[i][j][k][t+1];


}
    }
   }
    Eix[l][t+1]=s1;
    Bix[l][t+1]=s3;
}

//y component
for(int l=0;l<500;l++){
    for(int i=0;i<5;i++){
   for (int j=0;j<5;j++){
    for (int k=0;k<5;k++){

m1+=Ey[i][j][k][t+1];
m2+=By[i][j][k][t+1];


}
    }
   }
```

```
    Eey[l][t+1]=m1;
    Bey[l][t+1]=m2;
}
for(int l=0;l<500;l++){
    for(int i=0;i<5;i++){
   for (int j=0;j<5;j++){
    for (int k=0;k<5;k++){

m3+=Ey[i][j][k][t+1];
m4+=By[i][j][k][t+1];


}
    }
   }
    Eiy[l][t+1]=m3;
    Biy[l][t+1]=m4;
}

//z component
for(int l=0;l<500;l++){
    for(int i=0;i<5;i++){
   for (int j=0;j<5;j++){
    for (int k=0;k<5;k++){

n1+=Ez[i][j][k][t+1];
n2+=Bz[i][j][k][t+1];


}
    }
   }
    Eez[l][t+1]=n1;
    Bez[l][t+1]=n2;
}
for(int l=0;l<500;l++){
    for(int i=0;i<5;i++){
   for (int j=0;j<5;j++){
    for (int k=0;k<5;k++){

n3+=Ez[i][j][k][t+1];
n4+=Bz[i][j][k][t+1];

}
    }
   }
    Eiz[l][t+1]=n3;
    Biz[l][t+1]=n4;
}
string filename;
ofstream myfile;

stringstream d;
d<<t;
filename= "PSC_"+ d.str();
filename+= ".csv";
myfile.open(filename.c_str());
myfile<<"x,y,z,v,B,E\n";
cout<<"the results are :\n";
for(int i=0;i<500;i++){
```

```
myfile<<xe[i][t]<<","<<ye[i][t]<<","<<ze[i][t]<<","<<sqrt(pow(vex[i][t],2)+pow(v
ey[i][t],2)+pow(vez[i][t],2))<<","<<sqrt(pow(Bex[i][t],2)+pow(Bey[i][t],2)+pow(B
ez[i][t],2))<<","<<sqrt(pow(Eex[i][t],2)+pow(Eey[i][t],2)+pow(Eez[i][t],2))<<"\n
";

myfile<<xi[i][t]<<","<<yi[i][t]<<","<<zi[i][t]<<","<<sqrt(pow(vix[i][t],2)+pow(v
iy[i][t],2)+pow(viz[i][t],2))<<","<<sqrt(pow(Bex[i][t],2)+pow(Bey[i][t],2)+pow(B
ez[i][t],2))<<","<<sqrt(pow(Eix[i][t],2)+pow(Eiy[i][t],2)+pow(Eiz[i][t],2))<<"\n
";

}

myfile.close();
system("pause");
// new time step


}


    return a.exec();
}
```

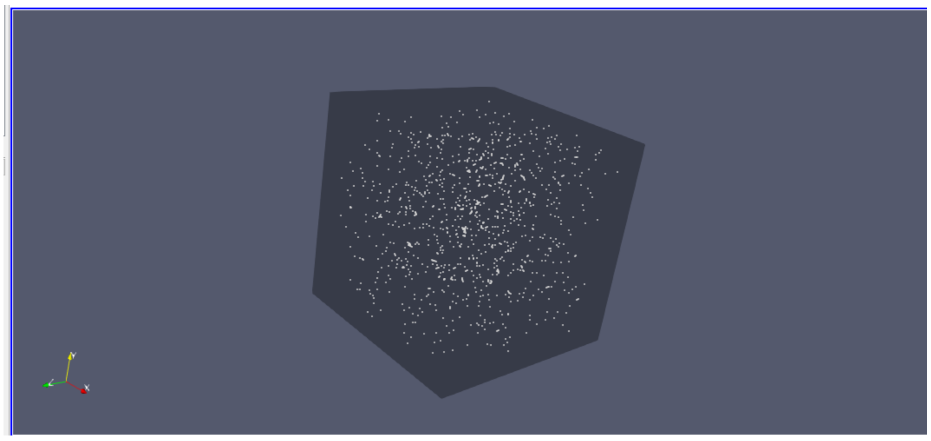
## 6.3 Results

The results are saved in 10 excel sheets and presented on Paraview.



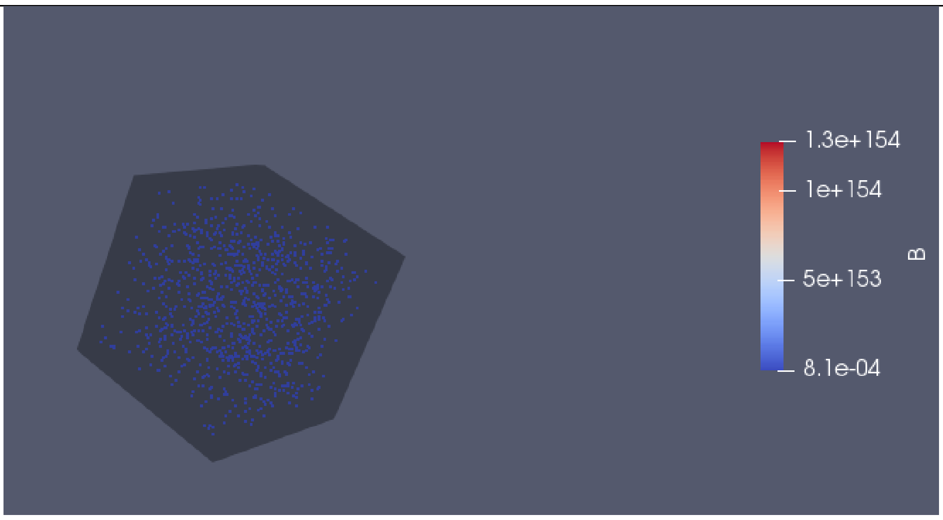
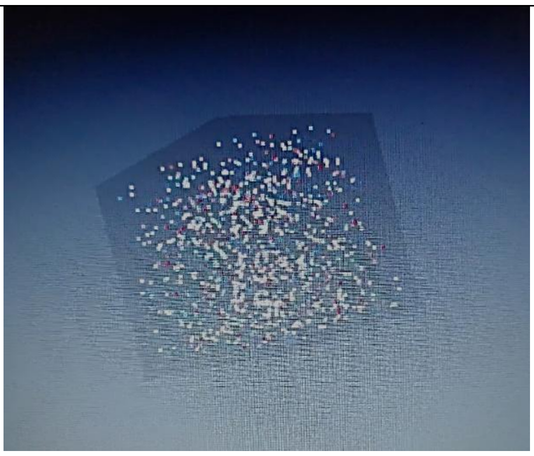
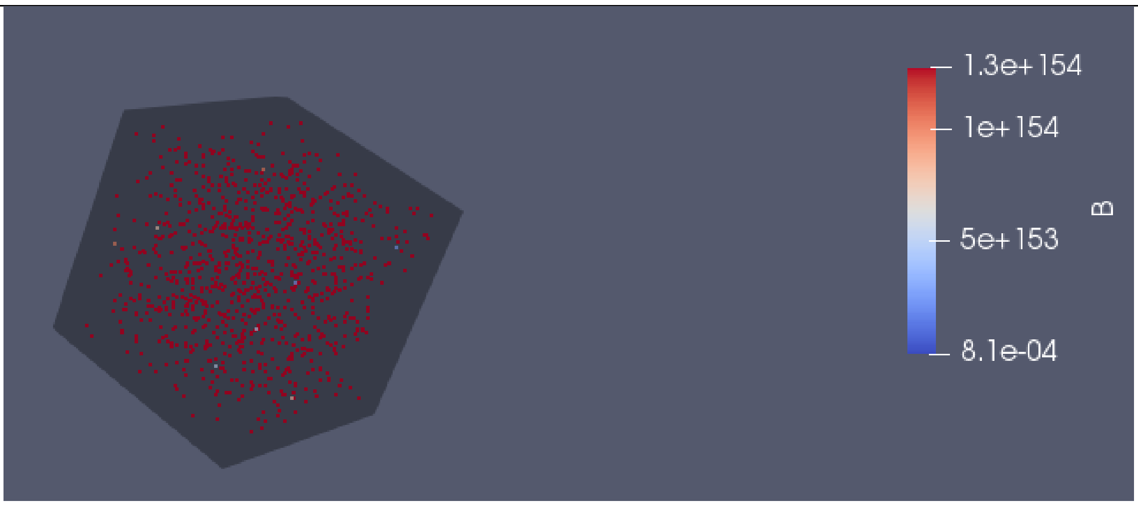
| | | | |
|---|---|---|---|
| PSC_0.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 39 KB |
| PSC_1.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 55 KB |
| PSC_2.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 49 KB |
| PSC_3.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 51 KB |
| PSC_4.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 51 KB |
| PSC_5.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 51 KB |
| PSC_6.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 51 KB |
| PSC_7.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 51 KB |
| PSC_8.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 27 KB |
| PSC_9.csv | 21-Aug-19 9:25 PM | Microsoft Excel C... | 25 KB |

The picture below presents the particles arbitrarily distributed in the box at t=0.



we can present any variable we want: velocity, electric field or magnetic field.

The pictures below present the variation of the **magnetic field** with time. The magnetic field varies from time step to another as we can see in the pictures.

| | |
|---|---|
|  | Time Step 1 |
|  | Time Step 5 |
|  | Time Step 9 |

## Literature

http://ptsg.egr.msu.edu