Chapter 3

The Simulation Code – XPDP1

3.1 Introduction

The simulation program used in this research is XPDP1, an open source software developed by the Plasma Theory and Simulation Group – PTSG at University of California, Berkeley. It is running on Unix workstations with X-Windows, and PC with an X-Windows emulator. XPDP1 is the X-Windows version of PDP1, which has the same physics kernel as PDP1. PDP1 is among the PDx1 codes from UC Berkeley, where "x" stands for P (Planar), C (Cylindrical), and S (Spherical) electrodes. The codes are written in object-oriented style, and in standard C (C language).

PDP1 (Plasma Device Planar) simulates a plasma within planar electrodes, with or without a uniform applied DC magnetic field in an arbitrary direction. The two electrodes in PDP1 are symmetric, as illustrated in Figure 3.1. For other versions in the same series, there are PDC1 and PDS1. PDC1 (Plasma Device Cylindrical) simulates a plasma within concentric (coaxial) cylindrical electrodes and allows an axial DC magnetic field. PDS1 (Plasma Device Spherical) simulates a plasma within concentric spherical electrodes, without a magnetic field. The inner electrode of both PDC1 and PDS1 is of finite size. They are useful for simulating discharges with different electrode areas [5]. In this work, we will only focus on the planar version, PDP1.

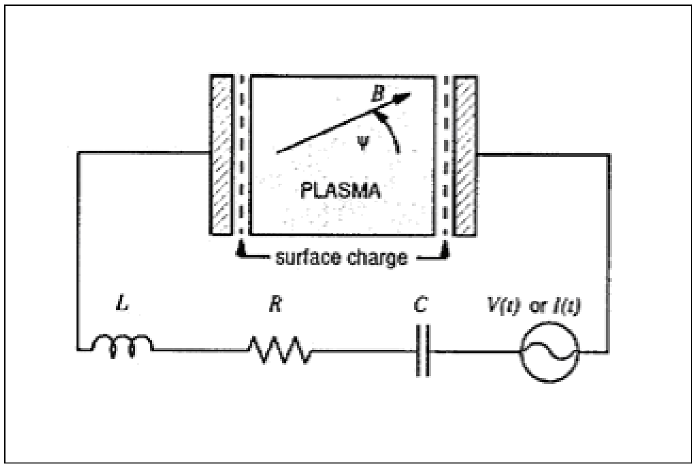


Figure 3.1: Schematic diagram of XPDP1 code [5].

PDP1 is the modified version of the W. S. Lawson's PDW1 code (1983). It is a bounded electrostatic code, simulating one-dimensional plasma devices. The code simulates a bounded plasma with external circuit, which include R, L, C elements, and AC, DC, ramped current/voltage sources. These characteristics (including particles and electrostatic fields) are specified by the user at run time using an input file. The code uses Particle-in-Cell (PIC) technique to simulate the electrons and ions, leap-frog method for the integration of the equation of motion, and Monte-Carlo collisional (MCC) model for electron-neutral and ion-neutral collisions [15]. The PIC technique, leap-frog method and MCC model are described in sections 3.2, 3.3 and 3.4 respectively. The simulation proceeds in real time, and the user may view the output as the code is running, in the form of various user-specified diagnostic windows. The diagnostic windows are updated at each time step (animation). The applications of XPDP1 code range from collisional capacitive RF discharges, used in materials processing to collisionless fusion problems [5].

3.3 Particle-in-Cell (PIC) Method

In reality, plasma is a collection of particles which consist of electrons, ions with various charge states, neutral atoms and molecules. In the Particle-in-Cell method, there are computer particles (superparticles). Each computer particle is a homogeneous collection of a large number of real particles (commonly 10^4 to 10^6 particles), which is having the same mass-to-charge ratio as the real particles, thus minimizing the amount of particles to be simulated.

In the PIC scheme, the physical volume is divided into cells by lines which run parallel to the boundaries. The intersections of these lines are called mesh points or grid points. An example of the grid is illustrated in Figure 3.2.

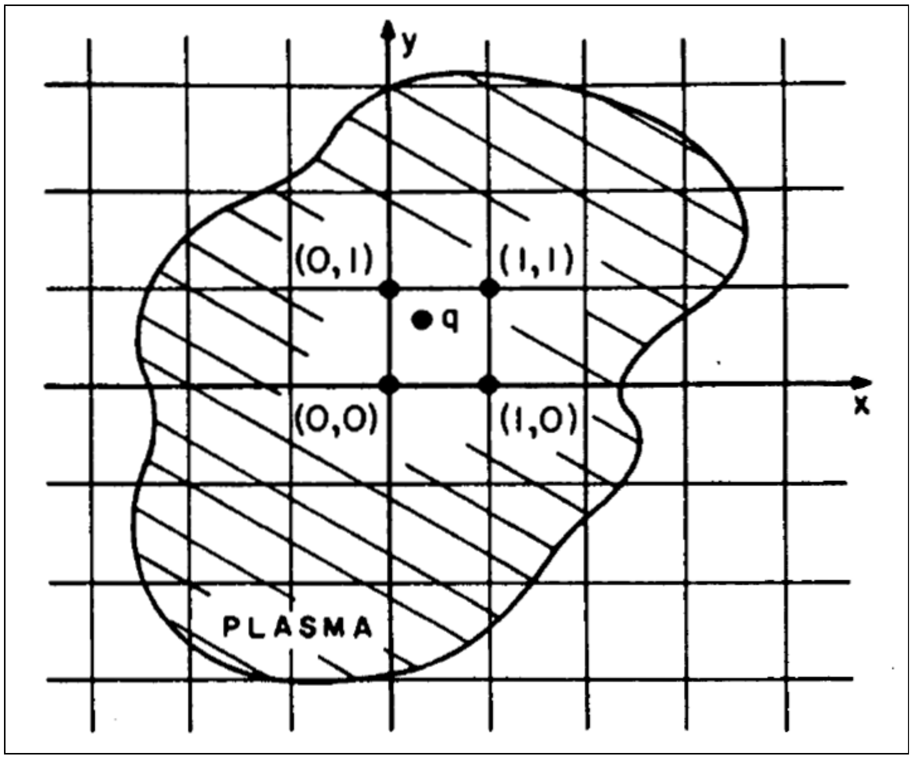


Figure 3.2: A mathematical grid of PIC scheme [16].

In Figure 3.2, A mathematical grid is set into the plasma region, to measure the charge density ρ and current density J. From the measured charge and current densities, we will obtain the electric field E and magnetic field B on the grid. The particle quantities such as velocity and position of a charged particle *q* at (x, y) location will be counted in terms of ρ at the nearby grid points (0, 0), (1, 0), (1, 1), (0, 1) and in terms of J at the faces between these points [16].

The particle quantities, such as velocities, **v** and position, **x** are known at the particle, and may take on all values in the phase space. For each time step, the charge and current densities on the grid are calculated. The process to produce the charge and current densities (ρ, **J**) on the grid from the particle positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$ implies some *weighting* (linear weighting) to the grid points that is dependent on particle position. Once the densities are established on the grid, the ρ and **J** are used to obtained the electric and magnetic fields (**E**, **B**) by solving the field equations. *Poisson's equation* is used to solve for **E** in electrostatic simulation. For electromagnetic simulations, the full set of *Maxwell's equations* is used to solve for **E** and **B**. With the fields on the grid, but particles scattered around within the grid, the force at the particle $\mathbf{F}_i$ is obtained through interpolation of the field from the grid to the particles by again performing a *weighting*. *Newton-Lorentz equation of motion* is used for the calculation, and the particles are advanced to new positions and velocities. Next, particle boundary conditions such as absorption and emission are applied. If the model is collisional, the Monte Carlo Collision (MCC) scheme is applied. The flow of the PIC-MCC scheme within one timestep is shown schematically in Figure 3.3 [16, 17, 18].

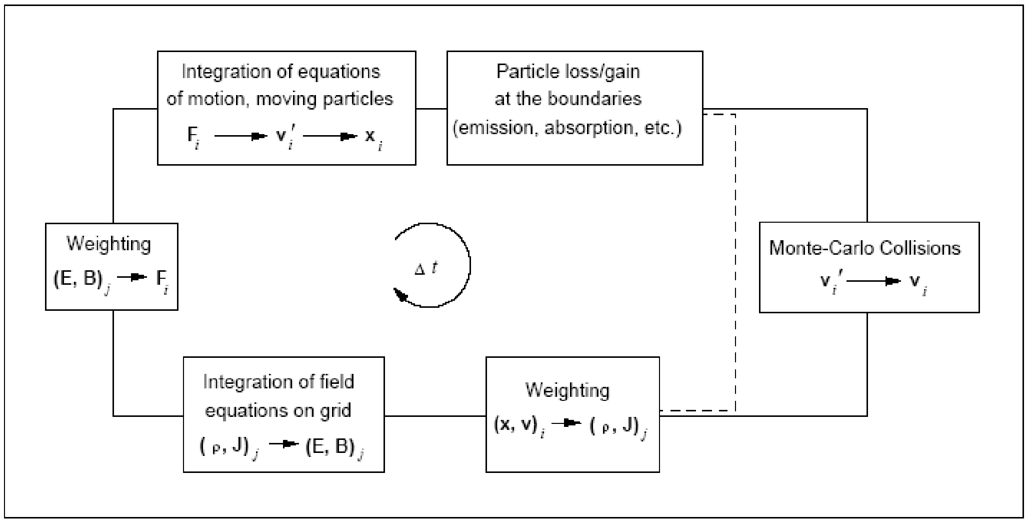

Integration of equations of motion, moving particles
$F_i \rightarrow v_i' \rightarrow x_i$
Particle loss/gain at the boundaries (emission, absorption, etc.)
Weighting
$(E, B)_j \rightarrow F_i$
Δt
Monte-Carlo Collisions
$v_i' \rightarrow v_i$
Integration of field equations on grid
$(\rho, J)_j \rightarrow (E, B)_j$
Weighting
$(x, v)_i \rightarrow (\rho, J)_j$

Figure 3.3: Flow chart for an explicit PIC-MCC scheme [16].

3.3 Integration of the Particle Equations of Motion

The integration method used in the code is called the *leap-frog* method. The two first-order differential equations to be integrated are

$$m\frac{dv}{dt} = F \ , \qquad \text{(3.1)}$$

$$\frac{dx}{dt} = v \ . \qquad \text{(3.2)}$$

These equations are replaced by the finite-difference equations

$$m\frac{v_{new} - v_{old}}{\Delta t} = F_{old} \ , \qquad \text{(3.3)}$$

$$\frac{x_{new} - x_{old}}{\Delta t} = v_{new} \ . \qquad \text{(3.4)}$$

In the *leap-frog* method, as shown in Figure 3.4, $v(0)$ is pushed back to $v(-\Delta t/2)$ using the force F calculated at $t = 0$. Consequently, the finite difference equations of the *leap-frog* method are

$$m\frac{v^{t+\Delta t/2} - v^{t-\Delta t/2}}{\Delta t} = F \ , \qquad \text{(3.5)}$$

$$\frac{x^{t+\Delta t} - x^{t}}{\Delta t} = v \ . \qquad \text{(3.6)}$$

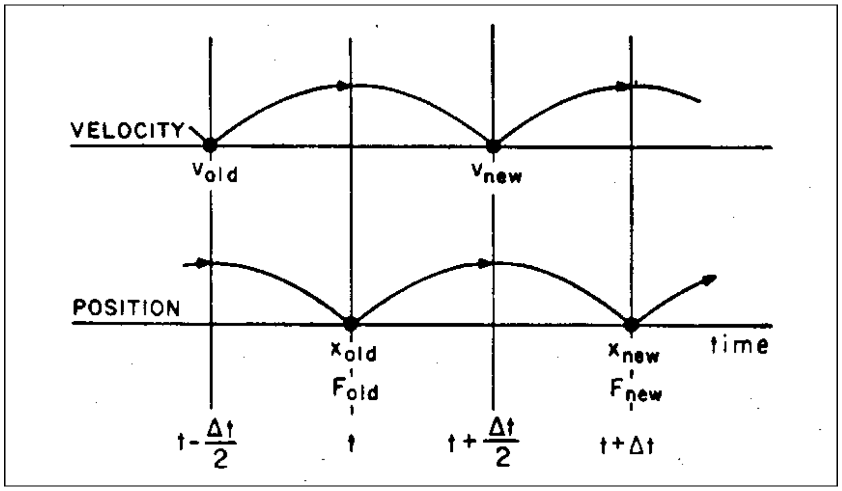


Figure 3.4: Scheme of the *leap-frog* integration method [4].

Therefore,

$$v^{t+\Delta t/2} = \left(\frac{F}{m}\times\Delta t\right) + v^{t-\Delta t/2} \ , \qquad \text{(3.7)}$$

$$x^{t+\Delta t} = v\times\Delta t + x^{t} \ . \qquad \text{(3.8)}$$

where force, $F = q(E + V\times B)$. The *leap-frog* method is "explicit" since $v^{t+\Delta t/2}$ and $x^{t+\Delta t}$ are determined only from values at earlier time levels. The stability and accuracy of the *leap-frog* method can be tested by applying it to the simple harmonic oscillator model

$$\frac{d^2x}{dt^2} = -\omega_0^2 x \,. \qquad \text{……..(3.9)}$$

Equation (3.9) is substituted into the finite difference scheme to obtain

$$\frac{x^{t+\Delta t} - 2x^t + x^{t-\Delta t}}{\Delta t^2} = -\omega_0^2 x^t \;. \qquad \text{…….(3.10)}$$

The solutions of the equation are of the form

$$x^t = C\exp(-i\omega t) \;, \qquad \text{…….(3.11)}$$

$$x^{t+\Delta t} = C\exp[-i\omega(t+\Delta t)] \;. \qquad \text{…...(3.12)}$$

Using Euler's formula, the finite difference becomes

$$\sin\left(\frac{\omega\Delta t}{2}\right) = \pm\frac{\omega_0\Delta t}{2} \;. \qquad \text{…….(3.13)}$$

For $\frac{\omega\Delta t}{2} << 1$, $\omega \approx \omega_0$ as desired. If $\omega_0\Delta t > 2$, the real solution for ω becomes complex with growing and decaying roots, indicating numerical instability. For simulations which use the leap-frog mover, typically $\omega_0\Delta t \leq 0.2$ is taken for stability [16, 17, 18]. More details on the accuracy and stability requirements for PIC are stated in section 3.5.

3.4 Monte Carlo Collision Model

In the Monte Carlo Collision (MCC) model, random numbers are being used to decide whether or not a particle is subjected to a collision, and what type of collision is to occur.

The MCC model statistically describes the collision processes, using cross sections for each type of collision. Consider a set of particles incident on another set of particles (targets). The probability, P_i of a collision event for the ith incident particle of energy $\varepsilon_i = \frac{1}{2} m v_i^2$ can be written as,

$$P_i = 1 - \exp[-n_g(x)\sigma_T(\varepsilon_i)v_i \Delta t] \,, \qquad \text{...(3.14)}$$

where the total cross section is the sum over all processes,

$$\sigma_T(\varepsilon_i) = \sum_j \sigma_j(\varepsilon_i) \,. \qquad \text{...(3.15)}$$

Here $n_g(x)$ is the spatially varying target density, v_i is the incident speed, and Δt is the time interval.

For example, assume the particle specie s has N types of collisions with the target specie, the kinetic energy of the ith particle of the incident s specie is given by

$$\varepsilon_i = \frac{1}{2} m_s v_i^2 \,, \qquad \text{...(3.16)}$$

where ε_i is needed in calculating the collision cross sections. The total cross section is $\sigma_T(\varepsilon_i) = \sum_j \sigma_j(\varepsilon_i)$, ……………………………………………………..(3.17)

where $\sigma_j(\varepsilon_i)$ for $1 \leq j \leq N$, is the cross section of the *j*th type of collision between the *s* specie and the target specie. The collision probability for the *i*th particle is calculated based on the distance $\Delta s_i = v_i \Delta t$ traveled in each time step, Δt . Consequently, $P_i = 1 - \exp[-n_g(x)\sigma_T(\varepsilon_i)v_i\Delta t] = 1 - \exp[-n_g(x)\sigma_T(\varepsilon_i)\Delta s_i]$. If a uniformly distributed random number on the related time step is less than P_i, a collision will take place. Then another random number is chosen to determine the type of collision. Every computer particle must be evaluated in every time step of the simulation.

Another approach is the *null collision* method, which is computationally more efficient. In this method, only one collision probability, which is energy independent, is used to model all the particles. The simulation program will randomly select a number of particles based on the collision probability, and concentrate on evaluating this selected fraction particles, then deciding which collision they are to undergo. The *null collision* model has a significant speed advantage over other collision models which query each of the fraction of particles for collisions at every time step.

3.5 Accuracy and Stability Requirements for PIC

The basic discretization parameters for "explicit" PIC method are mesh spacing, Δx, time step, Δt, and number of particles per cell, *PPC*. Trying to increase the mesh spacing, Δx and time step, Δt is often done for speeding-up the simulations. However, there are certain constraints on Δx and Δt in order to maintain accuracy and stability. For electrostatic PIC, the Δx and Δt are determined by

$$\omega_p \Delta t \le 0.2 \text{ and } \frac{\lambda_D}{\Delta x} \ge 1,$$

where the plasma frequency, $\omega_p = \left(\frac{e^2 n}{\varepsilon_0 m}\right)^{\frac{1}{2}}$, the Debye length, $\lambda_D = \left(\frac{\varepsilon_0 T}{en}\right)^{\frac{1}{2}}$ and the mesh spacing, $\Delta x = \frac{\textit{interelectrode gap length}}{\textit{number of spatial cells, nc}}$.

In these equations, T is the plasma temperature in electron volts, n is the plasma density, ε_0 is the permittivity of free space ($\approx 8.854\times10^{-12}$ F m^{-1}), e is the electronic charge, and m is the mass of the lightest species involved in the collision [18, 19].

3.6 Boundary Conditions

The boundary conditions of the code include surface charge on the electrodes, which are connected to a series RLC circuit with driving voltage, *V(t)* or current, *I(t)*. The general form of the applied source is

$$S(t) = DC + Ramp \cdot t + AC \cdot \sin(2\pi f_0 \cdot t + \theta_0)\,. \qquad \text{……………………………}(3.18)$$

An external circuit has its own intrinsic time scales which are not related to the plasma time scales. The simulation time step must be small enough to resolve these time scales, or the circuit simulation will produce inaccurate results regardless of the plasma parameters [20].

Referring to Figure 3.1, the current in the external circuit interacts with the plasma via the surface charge on the electrodes. The potential within the plasma region is affected by the distribution and motion of space charge, the electrode surface charge, and the current in the external circuit [21].

By applying Gauss' law to the system, the boundary conditions for the potential equation are obtained [21].

$$\oint_S E \cdot dS = \int_V \frac{\rho}{\varepsilon} dV + \frac{A_+\sigma_+ + A_-\sigma_-}{\varepsilon} = 0\ , \qquad \text{…………………………………......}(3.19)$$

where the surface S encloses the plasma and electrodes. A_+ and A_- refer to the left and right electrode respectively, and σ is the surface charge in the respective electrode. ρ has units of charge/volume and σ has units of charge/area. Using the definition of potential, we obtain

$$\frac{\Phi_{j+1} - 2\Phi_j + \Phi_{j-1}}{\Delta x^2} = -\frac{\rho_j}{\varepsilon} , \qquad \text{(3.20)}$$

for planar electrodes [21].

For one dimensional system, the boundary conditions can be written as [21]

$$\Phi_{nc} = 0 \qquad \text{(3.21)}$$

and

$$E_0 = \frac{\sigma_+}{\varepsilon} . \qquad \text{(3.22)}$$

Equation (3.21) can be written at one-half grid cell from the boundary, in conjunction with a central difference applied to the definition of potential, which gives

$$E_{1/2} = \frac{\Phi_0 - \Phi_1}{\Delta x} = \frac{1}{\varepsilon}\left(\sigma_+ + \rho_0 \frac{\Delta x}{2} \right) , \qquad \text{(3.23)}$$

for planar electrodes [24].

The charge conservation equation at each wall,

$$A\Delta\sigma = Q_{conv} + \Delta Q \qquad \text{(3.24)}$$

becomes

$$\sigma^t = \sigma^{t-\Delta t} + \frac{Q^t_{conv} + Q^t - Q^{t-\Delta t}}{A} . \qquad \text{(3.25)}$$

where Q is the charge on one plate of the external circuit capacitor [21].

In the code, there are four cases which cover the full range of external circuit parameters [21]:

(1) <u>General Series RLC Circuit (Voltage-Driven Circuit)</u>

For the general voltage-driven series RLC circuit, the capacitor charge Q is advanced using Kirchhoff's voltage law,

$$L\frac{d^2Q}{dt^2}+R\frac{dQ}{dt}+\frac{Q}{c}=V(t)+\Phi_{nc}-\Phi_0 \;. \qquad \ldots\ldots\ldots\ldots\ldots\ldots(3.26)$$

where Φ_{nc} is the reference potential fixed at zero, which to be the potential of the grounded electrode, and Φ_0 is the potential at time zero.

One second order difference is

$$Q^t=\frac{V(t)+\Phi_{nc}^t-\Phi_0^t-K^t}{\alpha_0}\;, \qquad \ldots\ldots\ldots\ldots\ldots\ldots(3.27)$$

where

$$K^t=\alpha_1 Q^{t-\Delta t}+\alpha_2 Q^{t-2\Delta t}+\alpha_3 Q^{t-3\Delta t}+\alpha_4 Q^{t-4\Delta t} \qquad \ldots\ldots\ldots\ldots...(3.28)$$

$$\alpha_0=\frac{9}{4}\frac{L}{\Delta t^2}+\frac{3}{2}\frac{R}{\Delta t}+\frac{1}{C}$$

$$\alpha_1=-6\frac{L}{\Delta t^2}-2\frac{R}{\Delta t}$$

$$\alpha_2=\frac{11}{2}\frac{L}{\Delta t^2}+\frac{1}{2}\frac{R}{\Delta t}$$

$$\alpha_3=-2\frac{L}{\Delta t^2}$$

$$\alpha_4=\frac{1}{4}\frac{L}{\Delta t^2}$$

(2) Open Circuit

When C → 0. the impedance approaches infinity, therefore the external circuit becoming an open circuit. The potentials on the boundaries are floating, and the surface charges on the electrodes influence the potential as always, but the electrodes cannot exchange charge via external current. The charge conservation equation is

$$\sigma_{+} = \sigma_{+}^{t-\Delta t} + Q_{conv}^{t} \quad . \qquad \ldots\ldots(3.29)$$

(3) Short-Circuit

When R=L=0 and C → ∞, the external circuit is a short circuit, with

$$\Phi_0 - \Phi_{nc} = V(t) \quad . \qquad \ldots\ldots(3.30)$$

The short-circuit case is applied when

$$\frac{C}{\varepsilon A / l} > 10^5 \quad . \qquad \ldots\ldots(3.31)$$

(4) Current-Driven Circuit

In this case, an ideal current source is assumed to be driving the specified time-varying current I(t). The external circuit elements R, L, and C are ignored since an ideal current source is an open circuit.

The stability of the circuit with L → 0 is given by the characteristic equation

$$\xi^2 (3 + 2\Delta t / RC) - 4\xi + 1 = 0 \quad , \qquad \ldots\ldots(3.32)$$

where $|\xi| \le 1$ is required. The roots are

$$\xi = \frac{2 \pm \sqrt{1 - 2\Delta t / RC}}{3 + 2\Delta t / RC} \quad . \qquad \ldots\ldots(3.33)$$

3.7 Particle Conditions - Loading Initial Distributions

In order to calculate the particle energy distribution, the following equations are used [17]:

The Maxwell-Boltzmann distribution,

$$f(\varepsilon) = f_0 \exp\left(-\frac{\varepsilon}{kT}\right) \quad , \qquad \ldots\ldots(3.34)$$

$$\varepsilon = \frac{1}{2} m \upsilon_t^2 = \frac{3}{2} kT \quad . \qquad \ldots\ldots(3.35)$$

In one dimension, equations (3.34) and (3.35) becomes,

$$f(\upsilon_i) = f_{0i} \exp\left(-\frac{\upsilon_i^2}{\upsilon_{ti}^2}\right) \quad , \qquad \ldots\ldots(3.36)$$

$$\frac{1}{2} m \upsilon_{ti}^2 = \frac{1}{2} kT \quad . \qquad \ldots\ldots(3.37)$$

Invert the cumulative distribution function (3.38),

$$F(\upsilon_i) = \frac{\int_0^{\upsilon_i} \exp(-\upsilon^2 / \upsilon_t^2) d\upsilon}{\int_0^{\infty} \exp(-\upsilon^2 / \upsilon_t^2) d\upsilon} \quad . \qquad \ldots\ldots(3.38)$$

By using Box-Muller method [35], we choose two pseudo-random numbers $(-1 < v_1, v_2 \leq 1)$ discarding them if $R^2 = v_1^2 + v_2^2 > 1$. Then

$$\upsilon_1 = v_1 \sqrt{-\frac{\ln(R^2)}{R^2}}, \qquad \text{...(3.39)}$$

$$\upsilon_2 = v_2 \sqrt{-\frac{in(R^2)}{R^2}}. \qquad \text{...(3.40)}$$

Equation (3.38) must be inverted numerically for general cutoffs.

3.8 General Structure of the Code

XPDP1 code is implemented with an object-oriented structure in standard C in order to facilitate enhancements. The code is separated into a physics application and the windowing core. Therefore, new physics and diagnostics can be added without altering the windowing core. The interaction between wingraphics manager and the physics kernel is illustrated in Figure 3.5.

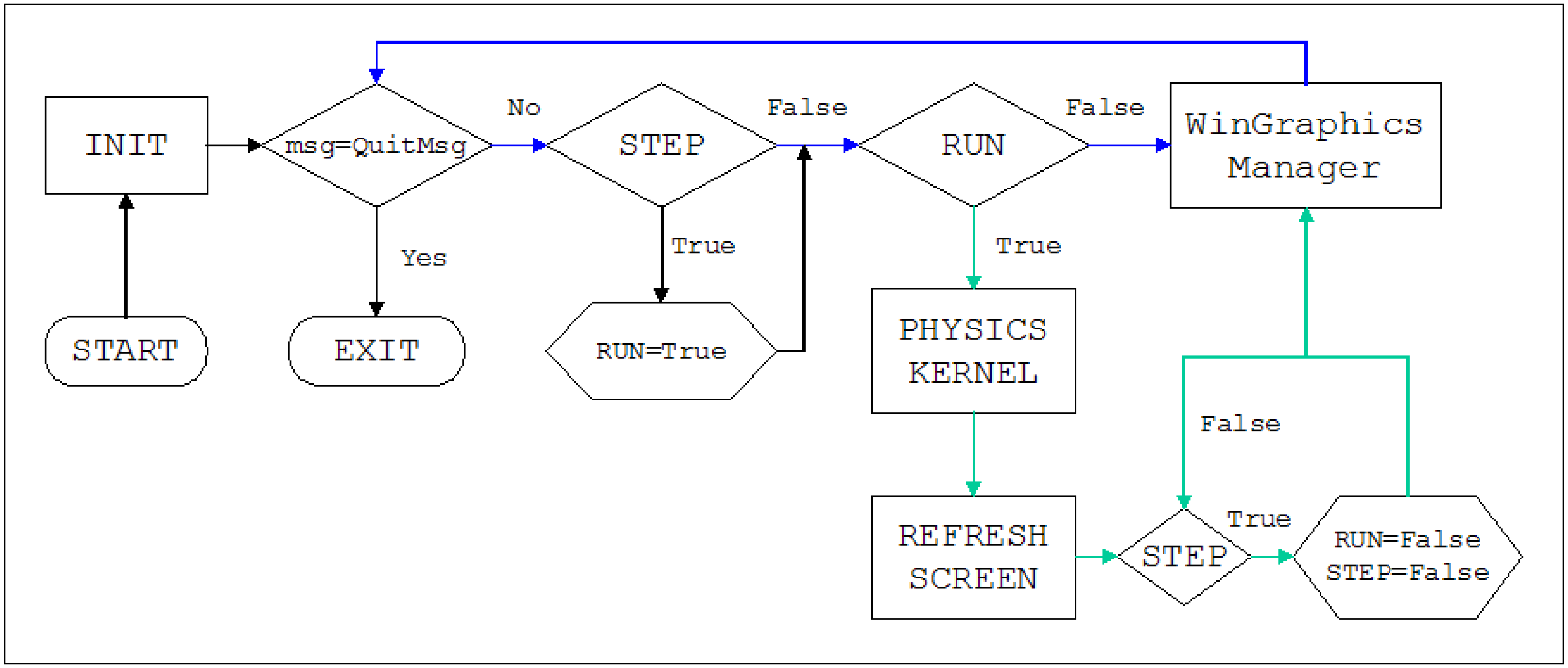


Figure 3.5: Schematic representation of the interaction between WinGraphics and the physics kernel [15].

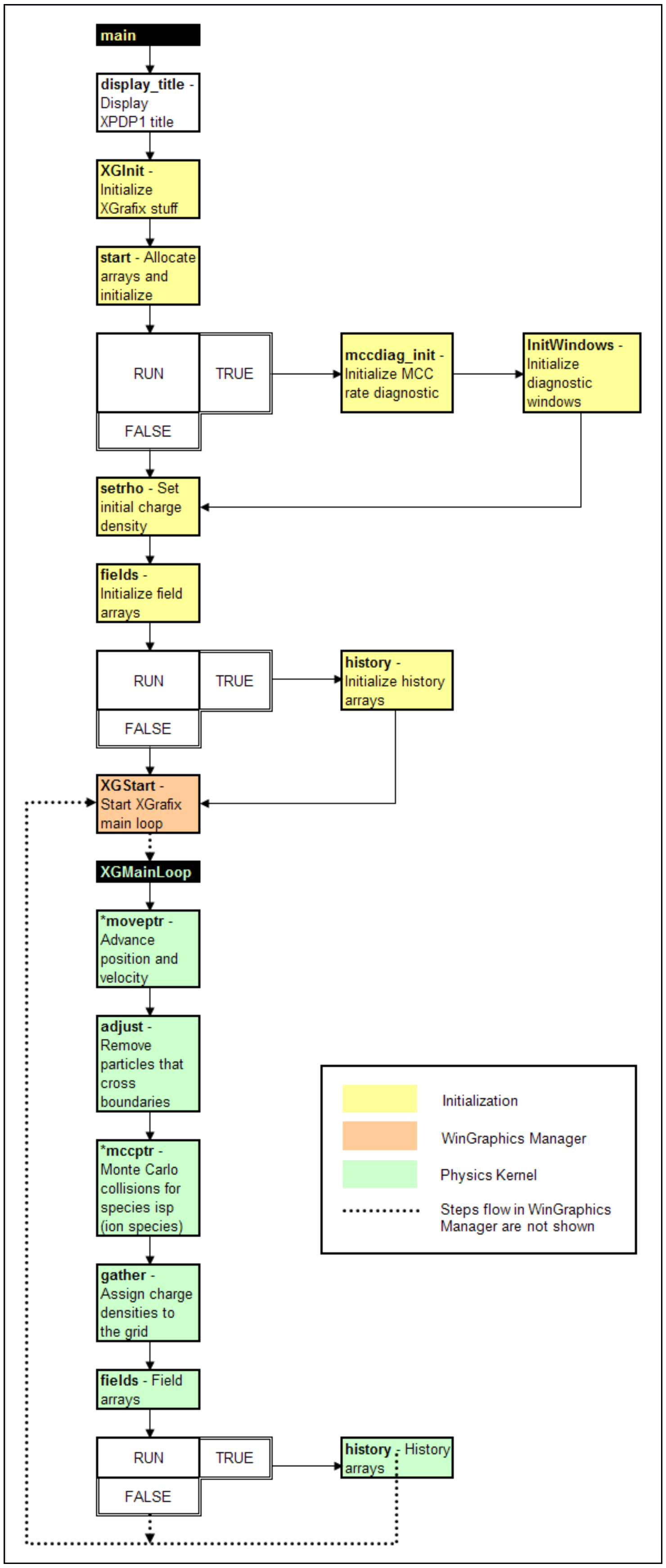
Using the windowing core, all diagnostics are updated dynamically in time. The diagnostics can also update in individual time-steps, and pausing for keystroke before continuing the simulation [15].

PDP1 can run indefinitely without a time step limitation. Time histories are combed periodically in such a way that there are never more than HISTMAX (a constant) value stored [15].

PDP1 employs four circuit solvers to handle the full range of external circuit parameters. (i) The general *voltage-driven series RLC* case. (ii) The *open circuit* case, where $C \rightarrow 0$. (iii) The *short circuit* case, when $C \rightarrow \infty$ and R=L=0. (iv) The *ideal current source* case. More details have been described previously in section 3.6.

A Monte Carlo Collisional model for electron-neutral and ion-neutral collisions is used in PDP1. All components of velocity can be specified independently in the input file.

The main flow of the code is shown in Figure 3.6. Table 3.1 is the description of the XPDP1 code main flow.



main
display_title - Display XPDP1 title
XGInit - Initialize XGrafix stuff
start - Allocate arrays and initialize
RUN
TRUE
FALSE
mccdiag_init - Initialize MCC rate diagnostic
InitWindows - Initialize diagnostic windows
setrho - Set initial charge density
fields - Initialize field arrays
RUN
TRUE
FALSE
history - Initialize history arrays
XGStart - Start XGrafix main loop
XGMainLoop
*moveptr - Advance position and velocity
adjust - Remove particles that cross boundaries
*mccptr - Monte Carlo collisions for species isp (ion species)
gather - Assign charge densities to the grid
fields - Field arrays
RUN
TRUE
FALSE
history - History arrays
Initialization
WinGraphics Manager
Physics Kernel
Steps flow in WinGraphics Manager are not shown

Figure 3.6: Main flow of XPDP1 code.

Table 3.1: Description of XPDP1 code main flow.

The description of the XPDP1 program main flow		
`display_title`	The code starts with the display of the program title and the general information of the program developer, etc..	
`XGInit`	Initialize the window graphics (XGrafix) stuff. Here, the window manager will get the code and the input file names, read, then initialize the window array.	
`Start`	The `start` function opens files and read the general input parameters in the input file, then checks for errors. The program will exit, when there is an error. If no error, the program will proceed to allocate field parameters and diagnostics arrays.	
	`DiagArray`	Allocate diagnostic arrays. This function will allocate space for all the diagnostics. The diagnostics are also group into time history diagnostics and average diagnostics.
	Then the program will setup the parameters for each species, e.g. ion, electron, etc..	
	`species`	Read species parameters from the input file, and assign input parameters to arguments.
	Scaling factors to convert from physical units to code unit and vis versa. Next, allocate particle arrays.	
	`SpeciesDiagArray`	This is the diagnostic arrays of the species. The program will allocate

		space for velocities distribution, and allocate space for the distribution functions arrays.
	Calculate the coefficient for the mover in the presence of a magnetic field. Then, the program will load in all species' particles, which is properly distributed. This will start with a "dump file" with all the record of a distribution, or if a dump file is not given, it will load the system with the initial distribution.	
	`Restore`	Running the dump file if given.
	`load`	Load one species and one direction at a time. There are loader for a cold beam and loader for thermal distribution. Maxwellian distribution function is used in the code. (More details in section 3.7 and Chapter 2 - section 2.6)
`mccdiag_init`	Initialize Monte Carlo Collision rate diagnostic. Allocate arrays for diagnostic rates.	
`InitWindows`	Initialize diagnostic windows. Setup each window structure.	
`setrho`	Set initial charge density, includes smoothing the charge density of each species.	
`fields`	Initialize field arrays.	
	`field_init`	Initializing the arrays and parameters for the field solve. Here, the program will setup the arrays for the poisson solve. Then, decide which circuit solver to use.

		There are four options of circuit solver: (More details in section 3.6) i. when $C \rightarrow 0$, open circuit ii. when current source is applied iii. when L=R=0, $C \rightarrow$infinity, short circuit iv. the general case with external voltage source	
	Calculate the external circuit.		
`History`	Initialize history arrays. This is the time history accumulator. It calculates and stores all history values, and performs combing on history values when low on memory.		
`XGStart`	This will start the XGrafix main loop, which is the wingraphics manager. It will then link to the main physics loop, which is the `XGMainLoop`.		
	`XGMainLoop`	For a no subcycling loop, initially the program will proceed to advance time for the species isp (ion). Then advance position and velocity in `(*moveptr)`.	
		`adjust`	Remove particles that cross boundaries. This is the routine to adjust (initialize, re-pack and inject) particles to the

			desired boundary conditions. There are four parts of simulation process. First, initialize array for computing positions of injected particles. Then, it will go to left hand side (LHS) wall diagnostics, mid system diagnostics, and injection of new particles at walls (one species at a time).
		`(*mccptr)` – e.g.: `argonmcc`	Monte Carlo Collisions for species isp (ion). This is the part of Monte Carlo Collision for electron-neutrals and ion-neutrals. Here, the program will be calculating the null collision probability, then the electron collisions with argon. For the collisions, there

			are five types of collisions: elastic collision, excitation, ionization, scattering and charge exchange collision. The program will determine the type of collision, and follow with the calculation. (More details in section 3.4 and Chapter 2 - section 2.7)
		`gather`	Assign charge densities to the grid. This function is in separate loop, because new particles might be created in `mcc` and `adjust` that might not have been weighted.
		Then the program will run the `fields`, and `history`. Lastly back to the flow in `XGStart` in the wingraphics manager.	

XPDP1 is available in the zipped form from the PTSG website at http://ptsg.eec.berkeley.edu [22]. The wingraphics manager (XGrafix) is available separately at the same website. The installation guide is available in the website at http://ptsg.eecs.berkeley.edu/~jhammel/qdptsg.html [23].

XPDP1 fully support a mouse for selection of items, buttons, etc.. The buttons on the main menu include RUN, STOP, STEP, SAVE, and QUIT. The diagnostic window has four buttons, which are RESCALE, TRACE, PRINT, and CROSS-HAIR.

The input file is used to specify the parameters for the simulation. The description of the input file is provided in the user manual [15], which comes together with the XPDP1 distribution.

There are “implicit” and “explicit” schemes in XPDP1. In this work, “explicit” scheme is used for all the simulations due to its simplicity as we are dealing with DC discharges. For the RF discharges we continue to use “explicit” scheme for comparison.