6 The Particle-in-Cell Method

David Tskhakaya

Association Euratom-OEAW, Institute of Theoretical Physics, A-6020 Innsbruck, Austria
Andronikashvili Institute of Physics, 380077 Tbilisi, Georgia

Probably the first Particle-in-Cell (PIC) simulations have been made at late 1950s by Buneman [1] and Dawson [2] who simulated the motion of 100–1 000 particles including interaction between them. Our day PIC codes simulate 10^5–10^{10} particles and represent a powerful tool for kinetic plasma studies. They are used practically in all branches of plasma physics modelling laboratory, as well as astrophysical plasma. PIC codes have a number of advantages: They represent so-called lowest codes, i.e. the number of assumptions made in the physical model is reduced to the minimum, they can simulate high-dimensional cases and can tackle complicated atomic and plasma-surface interactions. The prize for these advantages is a long simulation time: Some simulations can take up to 10^4 hours of CPU. As a result, they require a high level of optimization and are usually designed for professional use.

With this chapter we aim at introducing the reader to the basics of the PIC simulation technique. It is based mainly on available literature cited below, but includes some original unpublished material, too. For the interested reader I can recommend two classical monographs, [3] and [4], and the papers [5, 6] describing new developments in this field (see also references cited in the text).

The chapter is organized as follows. The main PIC features are discussed in Sect. 6.1. In Sect. 6.2 we consider solvers of equations of motion used in PIC and discuss their accuracy and stability aspects. Initialization of particle distribution, boundary effects and particle sources are described in Sect. 6.3. In Sects. 6.4 and 6.5 we show how plasma macro-parameters are calculated and discuss solvers of Maxwell's equations. Particle collisions are considered in Sect. 6.6. Final remarks are given in Sect. 6.7.

6.1 General Remarks

The idea of the PIC simulation is trivial: The code simulates the motion of plasma particles and calculates all macro-quantities (like density, current density and so on) from the position and velocity of these particles. The macro-force acting on the particles is calculated from the field equations. The name "Particle-in-Cell" comes from the way of assigning macro-quantities to the simulation particles. In general, any numerical simulation model, which simultaneously solves equations of motion of N particles

D. Tskhakaya: *The Particle-in-Cell Method*, Lect. Notes Phys. **739**, 161–189 (2008)
DOI 10.1007/978-3-540-74686-7_6 © Springer-Verlag Berlin Heidelberg 2008

$$\frac{\mathrm{d}\boldsymbol{X}_i}{\mathrm{d}t} = \boldsymbol{V}_i \qquad \text{and} \qquad \frac{\mathrm{d}\boldsymbol{V}_i}{\mathrm{d}t} = \boldsymbol{F}_i\left(t, \boldsymbol{X}_i, \boldsymbol{V}_i, A\right) \tag{6.1}$$

for $i = 1, \ldots, N$ and of macro fields $A = L_1(B)$, with the prescribed rule of calculation of macro quantities $B = L_2\left(\boldsymbol{X}_1, \boldsymbol{V}_1, \ldots, \boldsymbol{X}_N, \boldsymbol{V}_N\right)$ from the particle position and velocity can be called a PIC simulation. Here $\boldsymbol{X}_i$ and $\boldsymbol{V}_i$ are the generalized (multi-dimensional) coordinate and velocity of the particle i. A and B are macro fields acting on particles and some macro-quantities associated with particles, respectively. L_1 and L_2 are some operators and $\boldsymbol{F}_i$ is the force acting on a particle i. As one can see, PIC simulations have much broader applications then just plasma physics. On the other hand, inside the plasma community PIC codes are usually associated with codes solving the equation of motion of particles with the Newton-Lorentz's force (for simplicity we consider an unrelativistic case)

$$\frac{\mathrm{d}\boldsymbol{X}_i}{\mathrm{d}t} = \boldsymbol{V}_i \qquad \text{and} \qquad \frac{\mathrm{d}\boldsymbol{V}_i}{\mathrm{d}t} = \frac{e_i}{m_i}\left(\boldsymbol{E}\left(\boldsymbol{X}_i\right) + \boldsymbol{V}_i \times \boldsymbol{B}\left(\boldsymbol{X}_i\right)\right) \tag{6.2}$$

for $i = 1, \ldots, N$ and the Maxwell's equations

$$\begin{aligned} &\nabla \boldsymbol{D} = \rho\left(\boldsymbol{r}, t\right) , && \frac{\partial \boldsymbol{B}}{\partial t} = -\boldsymbol{\nabla} \times \boldsymbol{E} , && \boldsymbol{D} = \varepsilon \boldsymbol{E} , \\ &\nabla \boldsymbol{B} = 0 , && \frac{\partial \boldsymbol{D}}{\partial t} = \boldsymbol{\nabla} \times \boldsymbol{H} - \boldsymbol{J}\left(\boldsymbol{r}, t\right) , && \boldsymbol{B} = \mu \boldsymbol{H} , \end{aligned} \tag{6.3}$$

together with the prescribed rule of calculation of ρ and $\boldsymbol{J}$

$$\rho = \rho\left(\boldsymbol{X}_1, \boldsymbol{V}_1, \ldots, \boldsymbol{X}_N, \boldsymbol{V}_N\right) , \tag{6.4}$$

$$\boldsymbol{J} = \boldsymbol{J}\left(\boldsymbol{X}_1, \boldsymbol{V}_1, \ldots, \boldsymbol{X}_N, \boldsymbol{V}_N\right) . \tag{6.5}$$

Here ρ and $\boldsymbol{J}$ are the charge and current densities and ε and μ the permittivity and permeability of the medium, respectively. Below we will follow this definition of the PIC codes.

PIC codes usually are classified depending on dimensionality of the code and on the set of Maxwell's equations used. The codes solving a whole set of Maxwell's equations are called electromagnetic codes, contrary electrostatic ones solve just the Poisson equation. E.g., the XPDP1 code represents a 1D3V electrostatic code, which means that it is 1D in usual space and 3D in velocity space, and solves only the electrostatic field from the Poisson equation [7]. Some advanced codes are able to switch between different dimensionality and coordinate system, and use electrostatic, or electro-magnetic models (e.g. the XOOPIC code [8]).

A simplified scheme of the PIC simulation is given in Fig. 6.1. Below we consider each part of it separately.

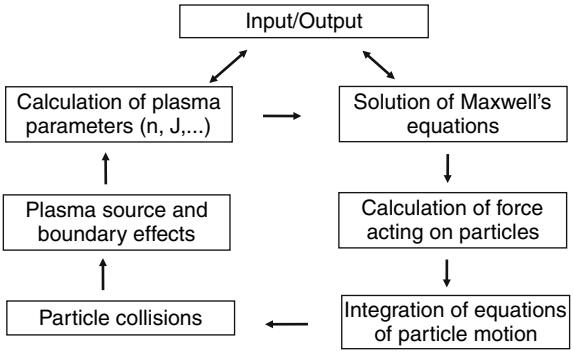

Fig. 6.1. Scheme of the PIC simulation

6.2 Integration of Equations of Particle Motion

6.2.1 Description of Particle Movers

During PIC simulation the trajectory of all particles is followed, which requires solution of the equations of motion for each of them. This part of the code is frequently called "particle mover".

A few words about the simulation particles itself. The number of particles in real plasma is extremely large and exceeds by orders of magnitude a maximum possible number of particles, which can be handled by the best supercomputers. Hence, during a PIC simulation it is usually assumed that one simulation particle consists of many physical particles. Because the ratio charge/mass is invariant to this transformation, this superparticle follows the same trajectory as the corresponding plasma particle. One has to note that for 1D and 2D models this transformation can be easily avoided by choosing of sufficiently small simulated volume, so that the number of real plasma particles can be chosen arbitrary.

As we will see below, the number of simulated particles is defined by a set of physical and numerical restrictions, and usually it is extremely large ($> 10^5$). As a result, the main requirements to the particle mover are the high accuracy and speed. One of such solvers represents the so called leap-frog method (see [3] and [4]), which we will consider in detail.

As in other numerical codes the time in PIC is divided into discrete time moments, in other words the time is grided. This means that physical quantities are calculated only at given time moments. Usually, the time step, Δt, between the nearest time moments is constant, so that the simulated time moments can be given via following expression: $t \to t_k = t_0 + k\Delta t$ and $A(t) \to A_k = A(t = t_k)$ with $k = 0, 1, 2, \ldots$, where t is the time, t_0 the initial moment and A denotes any physical quantity. The leap-frog method calculates particle velocity not at usual time steps t_k, but between them $t_{k+1/2} = t_0 + (k + 1/2)\,\Delta t$. In this way equations become time centred, so that they are sufficiently accurate and require relatively short calculation time

$$\frac{\boldsymbol{X}_{k+1}-\boldsymbol{X}_k}{\Delta t}=\boldsymbol{V}_{k+1/2}\ ,$$
$$\frac{\boldsymbol{V}_{k+1/2}-\boldsymbol{V}_{k-1/2}}{\Delta t}=\frac{e}{m}\left(\boldsymbol{E}_k+\frac{\boldsymbol{V}_{k+1/2}+\boldsymbol{V}_{k-1/2}}{2}\times\boldsymbol{B}_k\right)\ . \tag{6.6}$$

The leap-frog scheme is an explicit solver, i.e. it depends on old forces from the previous time step k. Contrary to implicit schemes, when for calculation of particle velocity a new filed (at time step $k+1$) is used, explicit solvers are simpler and faster, but their stability requires a smaller time step Δt.

By substituting

$$\boldsymbol{V}_{k\pm1/2}=\boldsymbol{V}_k\pm\frac{\Delta t}{2}\boldsymbol{V}'_k+\frac{\Delta t^2}{8}\boldsymbol{V}''_k\pm\frac{1}{6}\left(\frac{\Delta t}{2}\right)^3\boldsymbol{V}'''_k+\dots\ ,$$
$$\boldsymbol{X}_{k+1}=\boldsymbol{X}_k+\Delta t\boldsymbol{V}_k+\frac{\Delta t^2}{2}\boldsymbol{V}'_k+\frac{\Delta t^3}{6}\boldsymbol{V}''_k+... \tag{6.7}$$

into (6.6) we obtain the order of the error $\sim\Delta t^2$. It satisfies a general requirement for the scaling of numerical accuracy $\Delta t^{a>1}$. In order to understand this requirement we recall that for a fixed simulated time the number of simulated time steps scales as $N_t\sim\Delta t^{-1}$. Then, after N_t time steps an accumulated total error will scale as $N_t\Delta t^a\sim\Delta t^{a-1}$, where Δt^a is the scale of the error during one step. Thus, only $a>1$ can guarantee, that the accuracy increases with decreasing Δt.

There exist different methods of solution of finite-difference equations (see (6.6)). Below we consider the Boris method (see [3]), which is frequently used in PIC codes

$$\boldsymbol{X}_{k+1}=\boldsymbol{X}_k+\Delta t\boldsymbol{V}_{k+1/2}\qquad\text{and}\qquad\boldsymbol{V}_{k+1/2}=\boldsymbol{u}_++q\boldsymbol{E}_k \tag{6.8}$$

with $\boldsymbol{u}_+=\boldsymbol{u}_-+(\boldsymbol{u}_-+(\boldsymbol{u}_-\times\boldsymbol{h}))\times\boldsymbol{s}$, $\boldsymbol{u}_-=\boldsymbol{V}_{k-1/2}+q\boldsymbol{E}_k$, $\boldsymbol{h}=q\boldsymbol{B}_k$, $\boldsymbol{s}=2\boldsymbol{h}/(1+h^2)$ and $q=\Delta t/(2(e/m))$. Although these equations look very simple, their solution represent the most time consuming part of PIC, because it is done for each particle separately. As a result, the optimization of the particle mover can significantly reduce the simulation time.

In general, the Boris method requires 39 operations (18 adds and 21 multiplies), assuming that $\boldsymbol{B}$ is constant and $\boldsymbol{h}$, $\boldsymbol{s}$ and q are calculated only once at the beginning of simulation. But if $\boldsymbol{B}$ has one or two components, then the number of operations can be significantly reduced. E.g., if $\boldsymbol{B}\parallel\boldsymbol{z}$ and $\boldsymbol{E}\parallel\boldsymbol{x}$ then (6.8) can be reduced to the following ones

$$\boldsymbol{X}_{k+1}=\boldsymbol{X}_k+\Delta t\boldsymbol{V}_{k+1/2}\ ,$$
$$V^x_{k+1/2}=u^x_-+\left(V^y_{k+1/2}+V^y_{k-1/2}\right)h+qE^x_k\ ,$$
$$V^y_{k+1/2}=V^y_{k-1/2}\left(1-sh\right)-u^x_-s \tag{6.9}$$

with $u^x_-=V^x_{k-1/2}+qE^x_k$. They require just 17 operations (8 multiplies and 9 adds), which can save up to 50% of the CPU time. Some advanced PIC codes include a subroutine for searching the fastest solver for a given simulation setup, which significantly decreases the CPU time.

6.2.2 Accuracy and Stability of the Particle Mover

In order to find correct simulation parameters one has to know the absolute accuracy and corresponding stability conditions for the particle mover. They are different for different movers and the example considered below is applied just to the Boris scheme.

First of all let us consider the accuracy of a Larmor rotation. By assuming $\boldsymbol{V}_{k-1/2} \perp \boldsymbol{B}$ we can define the rotation angle during the time Δt from

$$\cos(\omega \Delta t) = \frac{\boldsymbol{V}_{k+1/2}\boldsymbol{V}_{k-1/2}}{V_{k-1/2}^2} \,. \tag{6.10}$$

On the other hand, substituting (6.8) into (6.10) we obtain

$$\frac{\boldsymbol{V}_{k+1/2}\boldsymbol{V}_{k-1/2}}{V_{k-1/2}^2} = \frac{1 - \frac{(\Delta t \Omega)^2}{4}}{1 + \frac{(\Delta t \Omega)^2}{4}} \tag{6.11}$$

with $\Omega = eB/m$, so that for a small Δt we get $\omega = \Omega(1 - (\Delta t \Omega)^2/12) + \ldots$. E.g., for a 1% accuracy the following condition has to be satisfied: $\Delta t \Omega \leq 0.35$.

In order to formulate the general stability condition some complicated calculations are required (see [4]). Below we present simple estimates of the stability criteria for the (explicit) particle mover.

Let us consider the equation of a linear harmonic oscillator

$$\frac{\mathrm{d}^2 X}{\mathrm{d}t^2} = -\omega_0^2 X \,, \tag{6.12}$$

having the following analytic solution

$$X = A\mathrm{e}^{-\mathrm{i}\omega_0 t} \,, \tag{6.13}$$

where A is an arbitrary imaginary number. The corresponding leap-frog equations take the following form

$$\frac{X_{k+1} - 2X_k + X_{k-1}}{\Delta t^2} = -\omega_0^2 X_k \,. \tag{6.14}$$

We assume that the solution has a form similar to (6.13), $X_k = A\exp(-\mathrm{i}\omega t_k)$. After substitution into (6.14) and performing simple transformations we find

$$\sin\left(\frac{\omega \Delta t}{2}\right) = \pm \frac{\omega_0 \Delta t}{2} \,. \tag{6.15}$$

Hence, for a stable solution $\mathrm{Im}(\omega) \leq 0$ the condition

$$\omega_0 \Delta t < 2 \tag{6.16}$$

is required. PIC often use a much more restrictive condition

$$\omega_0 \Delta t \leq 0.2 \ , \tag{6.17}$$

giving sufficiently accurate results. Interesting to note that this number has been derived few decades ago when the number of simulation time steps was typically of the order of $N_t \sim 10^4$. From (6.15) we obtain $\omega = \omega_0(1 - (\omega_0 \Delta t)^2/24) + \ldots$. Hence, a cumulative phase error after N_t steps should be $\Delta(\omega \Delta t) \approx (N_t(\omega_0 \Delta t)^3)/24$. Assuming $N_t = 10^4$ and $\Delta(\omega \Delta t) < \pi$ we obtain the condition (6.17). Although modern simulations contain much larger number of time steps up to $N_t = 10^7$, this condition still can work surprisingly well.

The restrictions on Δt described above can require the simulation of unacceptably large number of time steps. In order to avoid these restrictions different implicit schemes have been introduced: $\boldsymbol{V}_{k+1/2} = \boldsymbol{F}(\boldsymbol{E}_{k+1}, ...)$. The difference from the explicit scheme is that for the calculation of the velocity a new field is used, which is given at the next time moment.

One of examples of an implicit particle mover represents the so called 1 scheme (see [9])

$$\begin{aligned} \frac{\boldsymbol{X}_{k+1} - \boldsymbol{X}_k}{\Delta t} &= \boldsymbol{V}_{k+1/2} \ , \\ \frac{\boldsymbol{V}_{k+1/2} - \boldsymbol{V}_{k-1/2}}{\Delta t} &= \frac{e}{m} \left(\frac{\boldsymbol{E}_{k+1}(x_{k+1}) + \boldsymbol{E}_{k-1}}{2} \right. \\ &\quad \left. + \frac{\boldsymbol{V}_{k+1/2} + \boldsymbol{V}_{k-1/2}}{2} \times \boldsymbol{B}_k \right) \ . \end{aligned} \tag{6.18}$$

It can be shown that for a harmonic oscillator (see (6.12))

$$V, \ X \sim \frac{1}{(\omega_0 \Delta t)^{2/3}} \ , \tag{6.19}$$

if $\omega_0 \Delta t \gg 1$. Hence, the corresponding oscillations are heavily damped and the solver (see (6.18)) can filter unwanted oscillations. As a result, the condition (6.16) can be neglected.

6.3 Plasma Source and Boundary Effects

6.3.1 Boundary Effects

From the physics point of view, the boundary conditions for the simulated particles are relatively easy to formulate: Particles can be absorbed at boundaries, or injected from there with any distribution. On the other hand, an accurate numerical implementation of particle boundary conditions can be tricky. The problem is that (i) the velocity and position of particles are shifted in time ($\Delta t/2$), and (ii) the velocity of

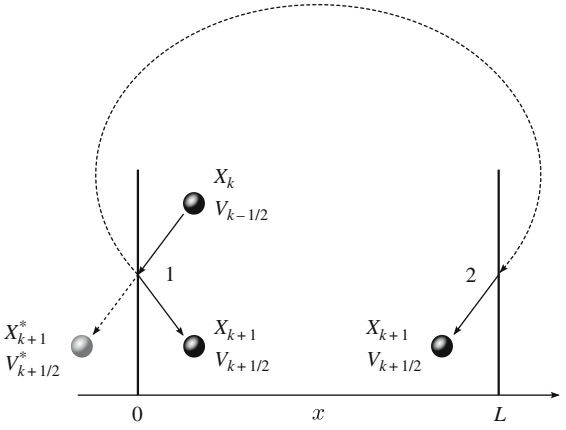

Fig. 6.2. Particle reflection (1) and reinjection (2) at the boundaries. X^*_{k+1} and $V^*_{k+1/2}$ denote the virtual position and velocity of a particle if there would be no boundary

particles are known at discrete time steps, while a particle can cross the boundary at any moment between these steps.

In unbounded plasma simulation particles are usually reflected at the boundaries, or reinjected from the opposite side (see Fig. 6.2). A frequently used reflection model, so called specular reflection, is given as

$$X^{\text{refl}}_{k+1} = -X_{k+1} \qquad \text{and} \qquad V^{x,\text{refl}}_{k+1/2} = -V^x_{k+1/2}\,. \tag{6.20}$$

Here, the boundary is assumed to be located at $x = 0$ (see Fig. 6.2). The specular reflection represents the simplest reflection model, but due to relatively low accuracy it can cause artificial effects. Let us estimate the accuracy of reflection (see (6.20)). The exact time when particle reaches a boundary and the corresponding velocity can be written as

$$\begin{aligned} t_0 &= t_k + \left| \frac{X_k}{V^x_{k-1/2}} \right| , \\ V_0 &= V^x_{k-1/2} + \left| \frac{X_k}{V^x_{k-1/2}} \right| \frac{e}{m} E^x_k\,. \end{aligned} \tag{6.21}$$

Accordingly, the velocity after the reflection will be

$$\begin{aligned} V^{x,\text{refl}}_{k+1/2} &= -V_0 + \left(\Delta t - \left| \frac{X_k}{V^x_{k-1/2}} \right| \right) \frac{e}{m} E^x_k \\ &= -V^x_{k-1/2} + \left(\Delta t - 2 \left| \frac{X_k}{V^x_{k-1/2}} \right| \right) \frac{e}{m} E^x_k\,. \end{aligned} \tag{6.22}$$

The second term on the right hand side of (6.22) represents the error made during the specular reflection, which can cause an artificial particle acceleration and heating.

Particle reinjection is applied usually when the fields satisfy periodic boundary conditions. The reinjection is given by $X_{k+1}^{\text{reinj}} = L - X_{k+1}$ and $V_{k+1/2}^{x,\text{reinj}} = V_{k+1/2}^{x}$, where $x = L$ denotes the opposite boundary. If the fields are not periodic, then this expression has to be modified. Otherwise a significant numerical error can arise.

The PIC codes simulating bounded plasmas are usually modeling particle absorption and injection at the wall, and some of them are able to tackle complicated plasma-surface interactions too.

Numerically, particle absorption is the most trivial operation and done by removing of the particle from memory. Contrary to this, for particle injection complicated numerical models can be required. When a new particle is injected it has to be taken into account that the initial coordinate and velocity are known at the same time, while the leap-frog scheme uses a time shifted values of them. In most cases the number of particles injected per time step is much smaller than the number of particles near the boundary, hence, the PIC code use simple injection models. For example, an old version of the XPDP1 code (see [7]) has used $\boldsymbol{V}_{k+1/2} = \boldsymbol{V} + e\Delta t\,(R - 0.5)\,\boldsymbol{E}_k/m$ and $X_{k+1} = R\Delta t V_{k+1/2}^{x}$, which assumes that particle has been injected at time $t_0 = t_{k+1} - R\Delta t$ with R being an uniformly distributed number between 0 and 1. $\boldsymbol{V}$ is the velocity obtained from a given injection distribution function (usually the Maxwellian one). The BIT1 code [10] uses a more simpler injection routine

$$\boldsymbol{V}_{k+1/2} = \boldsymbol{V} \qquad \text{and} \qquad X_{k+1} = R\Delta t V_{k+1/2}^{x}\,, \tag{6.23}$$

which is independent of the field at the boundary and hence, insensitive to a possible field error there. Description of higher order schemes can be found in [11].

Strictly speaking, the plasma-surface interaction processes can not be attributed to a classical PIC method, but probably all advanced PIC codes simulating bounded plasma contain elements of Monte-Carlo techniques [12]. A general scheme of plasma-surface interactions implemented in PIC codes is given below.

When a primary particle is absorbed at the wall, it can cause the emission of a secondary particle (a special case is reflection of the same particle). In general the emission probability F depends on the surface properties and primary particle energy ϵ and incidence angle α. Accordingly, the PIC code calculates $F\,(\epsilon, \alpha)$ and compares it to a random number R, uniformly distributed between 0 and 1. If $F > R$ then a secondary particle is injected. The velocity of a secondary particle is calculated according to a prescribed distribution $f_{\text{sev}}\,(\boldsymbol{V})$. Some codes allow multiple secondary particle injection, including as a special case the thermal emission. The functions F and f_{sev} are obtained from different sources on surface and solid physics.

6.3.2 Particle Loading

The particles in a PIC simulation appear, either by initial loading, or via particle injection from the boundary and at a volumetric source. In any case the corresponding velocities have to be calculated from a given distribution function $f\,(\boldsymbol{V})$. Important

to note, that there is a significant difference between volumetric particle loading and particle injection from the wall. In the first case the particle velocity is calculated directly from $f(\boldsymbol{V})$. Contrary to this, the velocity of particles injected from the wall has to be calculated according to $V^x f(\boldsymbol{V})$, where V^x is the component of the velocity normal to the wall. This becomes clear if we recall that the injection distribution function is the probability that particles having a distribution $f(\boldsymbol{V})$ will cross the boundary with a given velocity $\boldsymbol{V}$. For simplicity we do not distinguish below these two functions denoting them $f(\boldsymbol{V})$.

There exist two possibilities of calculation of velocities according to a given distribution:

(i) The most effective way for a 1D case is to use a cumulative distribution function

$$F(V) = \frac{\int\limits_{V_{\min}}^{V} f(V')\ \mathrm{d}\boldsymbol{V}'}{\int\limits_{V_{\min}}^{V_{\max}} f(V')\ \mathrm{d}\boldsymbol{V}'} \tag{6.24}$$

with $F(V_{\min}) = 0$ and $F(V_{\max}) = 1$, representing a probability that the velocity of particle lays between $V_{\min}$ and V. By equating this function to a sequence of uniformly distributed numbers U (or to random numbers R) between 0 and 1 and inverting it, we produce a sequence of V with the distribution $f(V)$ [3]:

$$F^{-1}(U) = V\ . \tag{6.25}$$

The same method can be applied to multi-dimensional cases which can be effectively reduced to 1D, e.g., by variable separation: $f(\boldsymbol{V}) = f_1(V^x)\, f_2(V^y)\, f_3(V^z)$. Often inversion of (6.25) can be done analytically, otherwise it is done numerically.

As an example we consider the injection of Maxwell-distributed particles: $f(V) \sim V \exp(-V^2/(2V_{\mathrm{T}}^2))$. According to (6.24) and (6.25) we get

$$F(V) = 1 - \mathrm{e}^{-V^2/(2V_{\mathrm{T}}^2)} \quad \text{and} \quad V = V_{\mathrm{T}}\sqrt{-2\ln(1-U)}\ . \tag{6.26}$$

(ii) Another possibility is to use two sets of random numbers R_1 and R_2 (for simplicity we consider a 1D case) $V = V_{\min} + R_1(V_{\max} - V_{\min})$, if $f(V)/(f_{\max}) > R_2$ use V, else try once more. This method requires random number generators of high level and it is time consuming. As a result, it is usually used when the method considered above can not be applied (e.g. for complicated multi-dimensional $f(\boldsymbol{V})$).

In advanced codes these distributions are generated and saved at the beginning of a simulation, so that later no further calculations are required except getting $\boldsymbol{V}$ from the memory. The same methods are used for spatial distributions $f(\boldsymbol{X})$, too.

As it was mentioned above, required velocity distributions can be generated by set of either ordered numbers U or by random numbers R, which are uniformly

distributed between 0 and 1. A proper choice of these numbers is not a trivial task and depends on the simulated system; e.g., using of random numbers can cause some noise. In addition, numerically generated random numbers in reality represent pseudo-random numbers, which can correlate and cause some unwanted effects. Contrary to this, the distributions generated by a set of ordered numbers, e.g. $U = (i + 0.5)/N$, $i = 1, \ldots, N-1$, are less noisy. On the other hand, in this case the generated distributions represent a multi-beam distribution, which sometimes can cause a beam instability [3].

6.4 Calculation of Plasma Parameters and Fields Acting on Particles

6.4.1 Particle Weighting

All numerical schemes considered up to now can be applied not only to PIC, but to any test particle simulation too. In order to simulate a real plasma one has to self-consistently obtain the force acting on particles, i.e. to calculate particle and current densities and solve Maxwell's equations. The part of the code calculating macro quantities associated with particles (n, $\boldsymbol{J}$, ...) is called "particle weighting".

For a numerical solution of field equations it is necessary to grid the space: $\boldsymbol{x} \rightarrow \boldsymbol{x_i}$ with $\boldsymbol{i} = 0, \ldots, \boldsymbol{N}_g$. Here $\boldsymbol{x}$ is a general 3D coordinate and $\boldsymbol{N}_g$ number of grid cells (e.g. for 3D Cartesian coordinates $\boldsymbol{N}_g = \left(N_g^x,\ N_g^y,\ N_g^z\right)$). Accordingly, the plasma parameters are known at these grid points: $A(\boldsymbol{x}) \rightarrow A_{\boldsymbol{i}} = A(\boldsymbol{x} = \boldsymbol{x}_i)$. The number of simulation particles at grid points is relatively low, so that one can not use an analytic approach of point particles, which is valid only when the number of these particles is very large. The solution is to associate macro parameters to each of the simulation particle. In other words to assume that particles have some shape $S(\boldsymbol{x} - \boldsymbol{X})$, where $\boldsymbol{X}$ and $\boldsymbol{x}$ denote the particle position and observation point. Accordingly, the distribution moments at the grid point $\boldsymbol{i}$ associated with the particle "j" can be defined as

$$A_{\boldsymbol{i}}^m = a_j^m S\left(\boldsymbol{x_i} - \boldsymbol{X}_j\right) \ , \tag{6.27}$$

where $A_{\boldsymbol{i}}^0 = n_{\boldsymbol{i}}$, $A_{\boldsymbol{i}}^1 = n_{\boldsymbol{i}} \boldsymbol{V}_{\boldsymbol{i}}$, $A_{\boldsymbol{i}}^2 = n_{\boldsymbol{i}} V_{\boldsymbol{i}}^2$ etc. and $a_j^0 = 1/V_g$, $a_j^1 = \boldsymbol{V}^j/V_g$, $a_j^2 = (V^j)^2/V_g$ etc. V_g is the volume occupied by the grid cell. The total distribution moments at a given grid point are expressed as

$$A_{\boldsymbol{i}}^m = \sum_{j=1}^{N} a_j^m S\left(\boldsymbol{x_i} - \boldsymbol{X}_j\right) \ . \tag{6.28}$$

Stability and simulation speed of PIC simulations strongly depend on the choice of the shape function $S(\boldsymbol{x})$. It has to satisfy a number of conditions. The first two conditions correspond to space isotropy

$$S(x) = S(-x) \ , \tag{6.29}$$

and charge conservation

$$\sum_{\boldsymbol{i}} S\left(\boldsymbol{x_i} - \boldsymbol{X}\right) = 1 \ . \tag{6.30}$$

The rest of the conditions can be obtained requiring an increasing accuracy of the weighting scheme. In order to derive them let us consider a potential generated at the point x by a unit charge located at the point X, $G\left(x - X\right)$. In other words $G\left(x - X\right)$ is the Green's function (for simplicity we consider a 1D case). Introducing the weighting scheme we can write the potential generated by some particle located at X as

$$\phi\left(x\right) = e \sum_{i=1}^{m} S\left(x_i - X\right) G\left(x - x_i\right) \ , \tag{6.31}$$

here e is the particle charge and m the number of nearest grid points with assigned charge. Expanding $G\left(x - x_i\right)$ near $\left(x - X\right)$ we get

$$\begin{aligned}
\phi\left(x\right) &= e \sum_{i=1}^{m} S\left(x_i - X\right) G\left(x - X\right) \\
&\quad + e \sum_{i=1}^{m} S\left(x_i - X\right) \sum_{n=1}^{\infty} \frac{\left(X - x_i\right)^n}{n!} \ \frac{\mathrm{d}^n G\left(x - X\right)}{\mathrm{d}x^n} \\
&= eG\left(x - X\right) + \delta\phi\left(x\right) \ , \\
\delta\phi\left(x\right) &= e \sum_{n=1}^{\infty} \frac{1}{n!} \ \frac{\mathrm{d}^n G\left(x - X\right)}{\mathrm{d}x^n} \sum_{i=1}^{m} S\left(x_i - X\right) \left(X - x_i\right)^n \ .
\end{aligned} \tag{6.32}$$

The first term on the right hand side of expression (6.32) represents a physical potential, while $\delta\phi$ is an unphysical part of it introduced by weighting. It is obvious to require this term to be as small as possible. This can be done by requiring

$$\sum_{i=1}^{m} S\left(x_i - X\right) \left(x_i - X\right)^n = 0 \tag{6.33}$$

with $n = 1, \ldots, n_{\max} - 1$. Substituting the expression (6.33) into (6.32) we get

$$\begin{aligned}
\delta\phi\left(x\right) &= \sum_{n=n_{\max}}^{\infty} \frac{1}{n!} \ \frac{\mathrm{d}^n G\left(x - X\right)}{\mathrm{d}x^n} \sum_{i=1}^{m} S\left(x_i - X\right) \left(X - x_i\right)^n \\
&\sim G\left(x - X\right) \sum_{i=1}^{m} S\left(x_i - X\right) \sum_{n=n_{\max}}^{\infty} \frac{\left(X - x_i\right)^n}{n! \left(x - X\right)^n} \ .
\end{aligned} \tag{6.34}$$

Thus, at large distance from the particle ($|X - x_i| < |x - X|$) $\delta\phi\left(x\right)$ decreases with increasing $n_{\max}$.

The shape functions can be directly constructed from the conditions (6.29), (6.30) and (6.33). The later two represent algebraic equations for $S\left(x_i - X\right)$. Hence, the number of conditions (6.33), which can be satisfied depends on the

maximum number of nearest grid points m to which the particle is weighted. A simplest shape function assigns density to the nearest grid point ($m = 1$) and satisfies just the first two conditions (6.29) and (6.30). For a 1D Cartesian coordinates it is given as $S^0(x) = 1$, if $|x| < \Delta x/2$, otherwise $S^0(x) = 0$, where Δx is the size of spatial grid. This weighting is called zero order or NGP weighting and was used in first PIC codes (see Fig. 6.3). Although the NGP scheme requires less CPU time, it is relatively noisy and probably not in use any more. The next, first order weighting scheme assigns density to two nearest grid points ($m = 2$) and given as $S^1(x) = 1 - |x|/(\Delta x)$, if $|x| < \Delta x$, otherwise $S^1(x) = 0$. Often it is called a cloud in cell (CIC) scheme. It satisfies one more condition in (6.33), with $n_{\max} = 1$, and a more accurate than the NGP scheme. Probably the CIC represents the most commonly used weighting scheme. Generalization to multi-dimensional Cartesian coordinates is trivial: $S(\boldsymbol{x}) = S(x)\,S(y)\,S(z)$. The higher order schemes (see [4]) can increase the accuracy of simulation (when other parameters are fixed), but require significantly longer CPU time.

For completeness I note, that some authors often use another definition of the particle shape $D(x)$ (e.g. see [4])

$$S(x_i - x) = \int_{x_i - \Delta x/2}^{x_i + \Delta x/2} D(x' - x)\ \mathrm{d}x' \,. \tag{6.35}$$

The meaning of this expression is that the density at the grid point x_i assigned by the particle located at the point x represents the average of the particle real shape $D(x' - x)$ over the area $[x_i - \Delta x/2; x_i + \Delta x/2]$. For the nearest grid point and

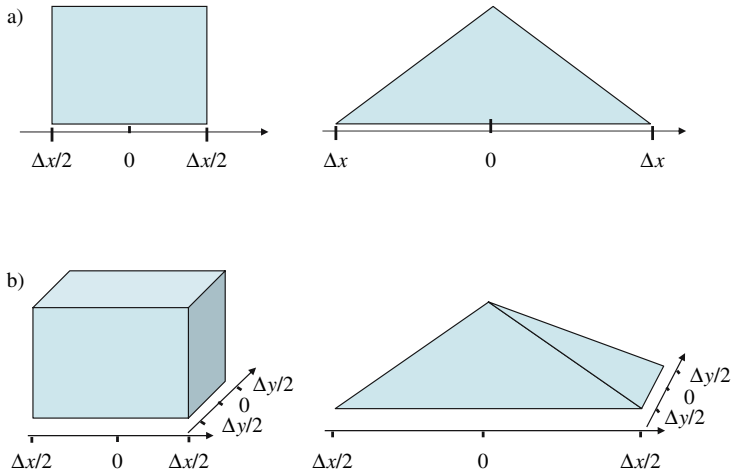

Fig. 6.3. Particle shapes for the NGP (**left**) and linear (**right**) weightings in 1D **(a)** and 2D **(b)** cases

linear weightings $D(x) = \delta(x)$ and $D(x) = H(\Delta x/2 - |x|)$, respectively. Here $H(x)$ is the step-function: $H(x) = 1$, if $x > 0$, else $H(x) = 0$.

6.4.2 Field Weighting

After the calculation of charge and current densities the code solves the Maxwell's equations (cf. Fig. 6.1) and delivers fields at the grid points $\boldsymbol{i} = 0, \ldots, \boldsymbol{N}_g$. These fields can not be used directly for the calculation of force acting on particles, which are located at any point and not necessarily at the grid points. Calculation of fields at any point is done in a similar way as charge assignment and called field weighting. So, we have $\boldsymbol{E_i}$ and $\boldsymbol{B_i}$ and want to calculate $\boldsymbol{E}(\boldsymbol{x})$ and $\boldsymbol{B}(\boldsymbol{x})$ at any point $\boldsymbol{x}$. This interpolation should conserve momentum, which can be done by requiring that the following conditions are satisfied:

(i) Weighting schemes for the field and particles are same

$$\boldsymbol{E}(\boldsymbol{x}) = \sum_{\boldsymbol{i}} \boldsymbol{E_i} \boldsymbol{S}(\boldsymbol{x_i} - \boldsymbol{x}) \ . \tag{6.36}$$

(ii) The field solver has a correct space symmetry, i.e. formally the field can be expressed in the following form (for simplicity we consider the 1D case)

$$E_i = \sum_k g_{ik} \rho_k \tag{6.37}$$

with $g_{ik} = -g_{ki}$, where ρ_k is the charge density at the grid point k. In order to understand this condition better, let us consider a 1D electrostatic system. By integrating the Poisson equation we obtain

$$E(x) = \frac{1}{2\varepsilon_0} \left(\int_a^x \rho \, \mathrm{d}x - \int_x^b \rho \, \mathrm{d}x \right) + E_b + E_a \ , \tag{6.38}$$

where a and b define boundaries of the system. Assuming that either a and b are sufficiently far and $E_{a,b} = \rho_{a,b} = 0$, or the system (potential) is periodic $E_b = -E_a$, $\rho_b = \rho_a$, we obtain

$$\begin{aligned} E(x_i) &= \frac{1}{2\varepsilon_0} \left(\int_a^{x_i} \rho \, \mathrm{d}x - \int_{x_i}^b \rho \, \mathrm{d}x \right) \\ &= \frac{\Delta x}{4\varepsilon_0} \left(\sum_{k=1}^{i-1} (\rho_k + \rho_{k+1}) - \sum_{k=i}^{N_g - 1} (\rho_k + \rho_{k+1}) \right) \\ &= \frac{\Delta x}{4\varepsilon_0} \sum_{k=1}^{N_g} g_{ik} \rho_k \end{aligned} \tag{6.39}$$

with $N_g \to \infty$, $\Delta x = [b,a]/N_g$ and

$$g_{ik} = \begin{cases} 2 & \text{if } i > k \\ -2 & \text{if } i < k \\ 0 & \text{if } i = k \end{cases} . \tag{6.40}$$

Thus, the condition (6.37) is satisfied.

Let us check different conservation constraints.

(i) The self-force of the particle located at the point x can be calculated as follows

$$\begin{aligned} F_{\text{self}} &= e\sum_i E_i S\left(x_i - x\right) = e\sum_{i,\,k} g_{ik} S\left(x_i - x\right)\rho_k \\ &= \frac{e^2}{V_g}\sum_{i,\,k} g_{ik} S\left(x_i - x\right) S\left(x_i - x\right) = \left(i \leftrightarrow k\right) \\ &= -\frac{e^2}{V_g}\sum_{i,\,k} g_{ik} S\left(x_i - x\right) S\left(x_i - x\right) = -F_{\text{self}} = 0 , \end{aligned} \tag{6.41}$$

(ii) The two-particle interaction force is given as

$$\begin{aligned} F_{12} &= e_1 E_2\left(x_1\right) = e_1\sum_i E_{2,i} S\left(x_i - x_1\right) \\ &= \frac{e_1 e_2}{V_g}\sum_{i,\,k} g_{ik} S\left(x_i - x_1\right) S\left(x_k - x_2\right) \\ &= -\frac{e_1 e_2}{V_g}\sum_{i,\,k} g_{ki} S\left(x_i - x_1\right) S\left(x_k - x_2\right) \\ &= -e_2\sum_{i,\,k} g_{ki} S\left(x_k - x_2\right)\rho_{1,k} = -e_2 E_1\left(x_2\right) \\ &= -F_{21} . \end{aligned} \tag{6.42}$$

Here, E_p denotes the electric field generated by the particle p.

(iii) Momentum conservation

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{P}}{\mathrm{d}t} &= \boldsymbol{F} = \sum_{p=1}^{N} e_p\left(\boldsymbol{E}\left(\boldsymbol{x}_p\right) + \boldsymbol{V}_p \times \boldsymbol{B}\left(\boldsymbol{x}_p\right)\right) \\ &= \sum_{p=1}^{N} e_p \sum_{\boldsymbol{i}} \boldsymbol{E_i} S\left(\boldsymbol{x}_i - \boldsymbol{x}_p\right) + \sum_{p=1}^{N} e_p \boldsymbol{V}_p \times \sum_{\boldsymbol{i}} \boldsymbol{B_i} S\left(\boldsymbol{x}_i - \boldsymbol{x}_p\right) \\ &= \sum_{\boldsymbol{i}} \boldsymbol{E_i} \sum_{p=1}^{N} e_p S\left(\boldsymbol{x}_i - \boldsymbol{x}_p\right) - \sum_{\boldsymbol{i}} \boldsymbol{B_i} \times \sum_{p=1}^{N} e_p \boldsymbol{V}_p S\left(\boldsymbol{x}_i - \boldsymbol{x}_p\right) \\ &= V_g \sum_{\boldsymbol{i}} \left(\rho_{\boldsymbol{i}} \boldsymbol{E_i} + \boldsymbol{J_i} \times \boldsymbol{B_i}\right) . \end{aligned} \tag{6.43}$$

Representing fields as a sum of external and internal components $\boldsymbol{E_i} = \boldsymbol{E_i^{\text{ext}}} + \boldsymbol{E_i^{\text{int}}}$ and $\boldsymbol{B_i} = \boldsymbol{B_i^{\text{ext}}}$, where $\boldsymbol{E_i^{\text{int}}}$ is given in expression (6.36), after some trivial transformations we finally obtain the equation of momentum conservation

$$\frac{\mathrm{d}\boldsymbol{P}}{\mathrm{d}t} = V_g \sum_{\boldsymbol{i}} \left(\rho_{\boldsymbol{i}} \boldsymbol{E_i^{\text{ext}}} + \boldsymbol{J_i} \times \boldsymbol{B_i} \right) . \tag{6.44}$$

As we see, the conditions (6.36) and (6.37) guarantee that during the force weighting the momentum is conserved and the inter-particle forces are calculated in a proper way. It has to be noted that:

(i) We neglected contribution of an internal magnetic field $\boldsymbol{B}^{\text{int}}$.
(ii) The momentum conserving schemes considered above does not necessarily conserve the energy too (for energy conserving schemes see [3] and [4]).
(iii) The condition (6.37) is not satisfied in general for coordinate systems with nonuniform grids, causing the self-force and incorrect inter-particle forces.

For exampple, if we introduce a nonuniform grid $\Delta x^i = \Delta x \alpha^i$ with $\alpha^i \neq \alpha^{j \neq i}$, in expression (6.39) we obtain

$$E\left(x_i\right) = \frac{\Delta x}{4\varepsilon_0} \sum_{k=1}^{N_g} g_{ik} \rho_k \tag{6.45}$$

with

$$g_{ik} = \begin{cases} \alpha^k + \alpha^{k-1} & \text{if } i > k \\ -\left(\alpha^k + \alpha^{k-1}\right) & \text{if } i < k, N_g \to \infty, \ \Delta x = \frac{[b,a]}{\sum_{i=1}^{N_g} \alpha^i} \\ \alpha^{i-1} - \alpha^i & \text{if } i = k \end{cases} , \tag{6.46}$$

so that $g_{ki} \neq -g_{ik}$.

6.5 Solution of Maxwell's Equations

6.5.1 General Remarks

Numerical solution of Maxwell's equations is a continuously developing independent direction in numerical plasma physics (e.g., see [13]). Field solvers in general can be divided into three groups:

(i) Mesh-relaxation methods, when the solution is initially guessed and then systematically adjusted until the solution is obtained with required accuracy;
(ii) Matrix methods, when Maxwell's equations are reduced to a set of linear finite difference equations and solved by some matrix method, and
(iii) Methods using the so called fast Fourier transform (FFT) and solving equations in Fourier space.

According to the type of the equations to be solved the field solvers can be explicit or implicit. E.g., the explicit solver of the Poisson equation solves the usual Poisson equation

$$\nabla \left[\varepsilon \left(\boldsymbol{x} \right) \nabla \varphi \left(\boldsymbol{x}, t \right) \right] = -\rho \left(\boldsymbol{x}, t \right) \ , \tag{6.47}$$

while an implicit one solves the following equation

$$\nabla \left[\left(1 + \eta \left(\boldsymbol{x} \right) \right) \varepsilon \left(\boldsymbol{x} \right) \varphi \left(\boldsymbol{x}, t \right) \right] = -\rho \left(\boldsymbol{x}, t \right) \ . \tag{6.48}$$

Here $\eta \left(\boldsymbol{x} \right)$ is the implicit numerical factor, which arises due to the fact that in its implicit formulation a new position (and hence ρ) of particle is calculated from a new field given at the same moment.

As an example we consider some matrix methods, which are frequently used in different codes. For a general overview of different solvers the interested reader can use [3] or [4].

6.5.2 Electrostatic Case, Solution of Poisson Equation

Let us consider the Poisson equation in a Cartesian coordinate system

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) \varphi \left(\boldsymbol{r} \right) = -\frac{1}{\varepsilon_0} \rho \left(\boldsymbol{r} \right) \ , \tag{6.49}$$

and formulate the corresponding finite difference equations. For this we use the transformation

$$\frac{\partial^2}{\partial x^2} \varphi \ \Rightarrow \ \frac{a\varphi_{i+1} + b\varphi_i + c\varphi_{i-1}}{\Delta x^2} \ . \tag{6.50}$$

Other components are treated in a similar way. Our aim is to choose the constants a, b and c, so that the error will be smallest. From the symmetry constraint we can write $a = c$. Then by expanding $\varphi_{i\pm1}$ at $x = x_i$

$$\begin{aligned} \varphi_{i\pm1} &= \varphi_i \pm \Delta x \left(\varphi_i \right)' + \frac{\Delta x^2}{2} \left(\varphi_i \right)'' \pm \frac{\Delta x^3}{6} \left(\varphi_i \right)''' + \frac{\Delta x^4}{24} \left(\varphi_i \right)^{(4)} \dots \ , \\ \left(\varphi_i \right)^{(k)} &= \left. \frac{\partial^k}{\partial x^k} \varphi \right|_{x=x_i} \ , \end{aligned} \tag{6.51}$$

and substituting in (6.50) we obtain

$$a\varphi_{i+1} + b\varphi_i + c\varphi_{i-1} = \varphi_i \left(2a + b \right) + \left(\varphi_i \right)'' a\Delta x^2 + \left(\varphi_i \right)^{(4)} a \frac{\Delta x^4}{12} + \dots \ . \tag{6.52}$$

Hence, by choosing $a = 1$ and $b = -2a = -2$ we get

$$\left(\frac{\partial^2 \varphi}{\partial x^2} \right)_{x=x_i} - \frac{\varphi_{i+1} - 2\varphi_i + \varphi_{i-1}}{\Delta x^2} = \frac{\Delta x^2}{12} \left(\varphi_i \right)^{(4)} + \mathcal{O} \left(\Delta x^4 \right) \ . \tag{6.53}$$

Hence, the finite difference equation (6.50) with $b = -2$ and $a = c = 1$ has second order accuracy ($\sim \Delta x^2$). Usually this accuracy is sufficient, otherwise one can consider a more accurate scheme

$$\frac{\partial^2}{\partial x^2}\varphi \;\Rightarrow\; \frac{a\varphi_{i+2} + b\varphi_{i+1} + c\varphi_i + d\varphi_{i-1} + e\varphi_{i-2}}{\Delta x^2} \,. \tag{6.54}$$

6.5.2.1 1D Case: Bounded Plasma with External Circuit

An excellent example of an 1D Poisson solver has been introduced in [7]. The solver is applied to an 1D bounded plasma between two electrodes and solves Poisson and external circuit equations simultaneously. Later, this solver has been applied to a 2D plasma model [14]. Below we consider an simplified version of this solver assuming that the external circuit consists of a voltage (or current) source $V(t)$ $(I(t))$ and a capacitor C (see Fig. 6.4)).

The Poisson equation for a 1D plasma is given as

$$\varphi_{i+1} - 2\varphi_i + \varphi_{i-1} = -\frac{\Delta x^2}{\varepsilon_0}\rho_i \,. \tag{6.55}$$

It is a second order equation, so that we need two boundary conditions for the solution. The first one can be a potential at the right-hand-side (rhs) wall:

$$\varphi_{Ng} = 0. \tag{6.56}$$

The second condition can be formulated at the left-hand-side (lhs) wall:

$$\frac{\varphi_0 - \varphi_1}{\Delta x} = E(x = \frac{\Delta x}{2}) = E_0 + \frac{1}{\varepsilon_0}\int_0^{\Delta x/2} \rho \, \mathrm{d}x \approx E_0 + \frac{\Delta x}{2\varepsilon_0}\rho_0 \,. \tag{6.57}$$

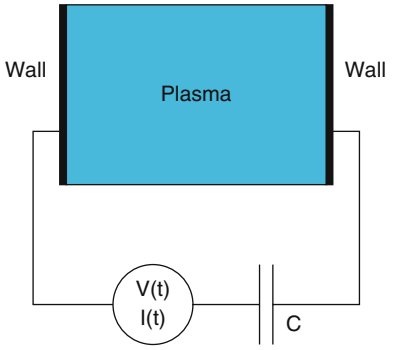

Fig. 6.4. Scheme of 1D bounded plasma with external circuit

Recalling that E_0 is the electric field at the l.h.s. wall, we can write $E_0 = \sigma_{\text{lhs}}/\varepsilon_0$, where σ_{lhs} is the surface charge density there. Hence, the second boundary condition can be formulated as

$$\varphi_0 - \varphi_1 = \frac{\Delta x}{\varepsilon_0}\left(\sigma_{\text{lhs}} + \frac{\Delta x}{2}\rho_0\right) . \tag{6.58}$$

In order to calculate σ_{lhs} we have to employ the circuit equation.

6.5.2.1.1 Voltage Driven Source with Finite C

In this case charge conservation at the l.h.s. wall can be written as

$$\sigma_{\text{lhs}}(t) = \sigma_{\text{lhs}}(t - \Delta t) + \frac{Q_{\text{pl}} + Q_{\text{ci}}}{S} , \tag{6.59}$$

where Q_{pl} and Q_{ci} are the charge deposited during Δt time on the lhs wall by the plasma and the external circuit, respectively. S is the area of the wall surface. Q_{pl} can be calculated by counting the charge of the plasma particles absorbed at the lhs wall, and Q_{ci} can be given as $Q_{\text{ci}} = Q^{\text{c}}(t) - Q^{\text{c}}(t - \Delta t)$, where Q^{c} is the charge at the capacitor. Q^{c} can be calculated using the Kirchhoff's law

$$\frac{Q^{\text{c}}}{C} = V(t) + \varphi_{N_g} - \varphi_0 = V(t) - \varphi_0 . \tag{6.60}$$

Substituting the expressions (6.59) and (6.60) into (6.58) we obtain

$$\begin{aligned} &\varphi_0\left(1 + \frac{C}{S}\frac{\Delta x}{\varepsilon_0}\right) - \varphi_1 \\ &= \frac{\Delta x}{\varepsilon_0}\left(\frac{Q_{\text{pl}} + C\left(V(t) - V(t - \Delta t) + \varphi_0(t - \Delta t)\right)}{S}\right. \\ &\quad \left. + \sigma_{\text{lhs}}(t - \Delta t) + \frac{\Delta x}{2}\rho_0\right) . \end{aligned} \tag{6.61}$$

6.5.2.1.2 Voltage Driven Source with $C \to \infty$

In this case in spite of (6.58) we use

$$\varphi_0 - \varphi_{Ng} = \varphi_0 = V(t) . \tag{6.62}$$

6.5.2.1.3 Open Circuit ($C = 0$)

In this case we write

$$\sigma_{\text{lhs}}(t) = \sigma_{\text{lhs}}(t - \Delta t) + \frac{Q_{\text{pl}}}{S} , \tag{6.63}$$

so that the second boundary condition takes the following form

$$\varphi_0 - \varphi_1 = \frac{\Delta x}{\varepsilon_0}\left(\sigma_{\text{lhs}}(t - \Delta t) + \frac{Q_{\text{pl}}}{S} + \frac{\Delta x}{2}\rho_0\right) . \tag{6.64}$$

6.5.2.1.4 Current Driven Source

In this case Q_{ci} can be directly calculate from the expression $Q_{ci} = \Delta t I(t)$. Then the second boundary condition can be given as

$$\varphi_0 - \varphi_1 = \frac{\Delta x}{\varepsilon_0}\left(\sigma_{\text{lhs}}\left(t - \Delta t\right) + \frac{Q_{\text{pl}} + \Delta t I\left(t\right)}{S} + \frac{\Delta x}{2}\rho_0\right) . \tag{6.65}$$

Combining equations (6.55), (6.56) and (6.61)–(6.65) we can write the set of difference equations in the following matrix form

$$\begin{pmatrix} a & b & 0 & \ldots\ldots\ldots & 0 \\ c & -2 & 1 & 0 & \ldots\ldots & 0 \\ 0 & 1 & -2 & 1 & 0 & \ldots & 0 \\ & & \ddots & \ddots & \ddots & & \\ 0 & \ldots & 0 & 1 & -2 & 1 & 0 \\ 0 & \ldots\ldots & 0 & 1 & -2 & 1 & \\ 0 & \ldots\ldots\ldots & 0 & 1 & -2 & & \end{pmatrix} \begin{pmatrix} \varphi_0 \\ \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_{Ng-2} \\ \varphi_{Ng-1} \end{pmatrix} = -\frac{\Delta x^2}{\varepsilon_0} \begin{pmatrix} d/\Delta x \\ \rho_1 + e \\ \rho_2 \\ \vdots \\ \rho_{Ng-2} \\ \rho_{Ng-1} \end{pmatrix} . \tag{6.66}$$

Here, for the cases

(i) Voltage driven source or open circuit: $a = -1 - C\Delta x/S\varepsilon_0$, $b = 1$, $c = 1$, $d = \sigma_{\text{lhs}}(t - \Delta t) + (Q_{\text{pl}} + C(V(t) - V(t - \Delta t) + \varphi_0(t - \Delta t)))/S + \Delta x/2\rho^0$ and $e = 0$.
(ii) Short circuit ($C \to \infty$): $a = b = c = d = 0$ and $e = \varepsilon_0/(\Delta x^2)V\left(t\right)$.
(iii) Current driven source: $a = -1$, $b = 1$, $c = 1$, $d = \sigma_{\text{lhs}}(t - \Delta t) + (Q_{\text{pl}} + \Delta t I(t))/S + \Delta x/2\rho^0$ and $e = 0$.

The matrix (6.66) can be solved by standard inverse matrix solvers (e.g., see [15]).

6.5.2.2 2D Case: Generalization of the 1D Solver

This 1D solver can be generalized for a 2D case. The main difference between the 1D and 2D cases represent the decomposition of the field and the boundary conditions at internal objects introduced in 2D (for details see [14]).

Field decomposition is given by

$$\varphi\left(t, x, y\right) = \varphi^{\text{pl}}\left(t, x, y\right) + \varphi^{\text{con}}\left(t\right)\varphi^{\text{vac}}\left(x, y\right) . \tag{6.67}$$

Here φ^{pl} is the plasma field with the zero boundary conditions

$$\Delta\varphi^{\text{pl}}\left(t, x, y\right) = -\frac{1}{\varepsilon_0}\rho\left(x, y\right) , \qquad \varphi^{\text{pl}}\big|_b = 0 , \tag{6.68}$$

where φ^{vac} is the vacuum field with the unit boundary conditions

$$\Delta\varphi^{\text{vac}}(x,y) = 0\,, \qquad \varphi^{\text{vac}}|_b = 1\,. \tag{6.69}$$

$\varphi^{\text{con}}(t)$ is the field at conductors, which is either calculated self-consistently (for electrodes), or prescribed (e.g., at the wall). The symbol $|_b$ denotes a plasma boundary.

It's easy to see that φ in (6.67) represents an exact solution of the Poisson equation with the given boundary conditions. The advantage of this decomposition is that

(i) the vacuum field has to be calculated just once and
(ii) the Poisson equation (6.68) with the zero boundary conditions is easier to solve, than one with a general boundary conditions.

As a result, the field decomposition can save a lot of CPU time.

The equation of the plasma field (6.68) is reduced to a set of finite difference equations

$$\frac{\varphi_{i+1,j} - 2\varphi_{ij} + \varphi_{i-1,j}}{\Delta x^2} + \frac{\varphi_{i,j+1} - 2\varphi_{ij} + \varphi_{i,j-1}}{\Delta y^2} = -\frac{1}{\varepsilon_0}\rho_{ij} \tag{6.70}$$

with $\varphi|_b = 0$, which can be solved by matrix method. In a similar way the Laplace equation (6.69) can be solved for the vacuum field.

The corresponding boundary conditions at the wall of internal objects are calculated using Gauss' law

$$\oint \varepsilon \boldsymbol{E}\, \mathrm{d}\boldsymbol{S} = \int \rho\, \mathrm{d}V + \oint \sigma\, \mathrm{d}S\,, \tag{6.71}$$

which in a finite volume representation can be written as

$$\begin{aligned} &\Delta y \Delta z \left(\varepsilon_{i+1/2,j} E_{i+1/2,j} - \varepsilon_{i-1/2,j} E_{i-1/2,j}\right) \\ &\qquad + \Delta x \Delta z \left(\varepsilon_{i,j+1/2} E_{i,j+1/2} - \varepsilon_{i,j-1/2} E_{i,j-1/2}\right) \\ &= \rho_{ij} \Delta V_{ij} + \sigma_{ij} \Delta S_{ij}\,. \end{aligned} \tag{6.72}$$

Here ΔV_{ij} and ΔS_{ij} are the volume and area associated with the given grid point i, j. The electric fields entering in this equation are calculated according to the following expressions

$$\begin{aligned} E_{i\pm1/2,j} &= \pm\frac{\varphi_{i,j} - \varphi_{i\pm1,j}}{\Delta x}\,, \\ E_{i,j\pm1/2} &= \pm\frac{\varphi_{i,j} - \varphi_{i,j\pm1}}{\Delta x}\,. \end{aligned} \tag{6.73}$$

Calculation of the potential at the plasma boundary $\varphi^{\text{con}}(t)$ consists in general of three parts. The potential at the outer wall is fixed and usually chosen as 0. The potential at the electrodes, which are connected to an external circuit is done in a similar way as for the 1D case considered above. For calculation of the potential at the internal object equation (6.72) is solved. We note that the later task is case dependent and not a trivial one, e.g., the solution depends on the object shape or material (conductor or dielectric). For further details see [14].

6.5.2.3 2D Case: Cartesian/Fourier Solver

The number of operations to be performed by a matrix solver (per dimension) scales as $\sim N_g^2$ and drastically increases with N_g. This number can be significantly reduced by using a fast Fourier Transform (FFT) solver, which scales as $\sim N_g \ln N_g$ (see [15]). This scaling can be significantly improved by using different optimizations. One example when FFT solvers can be applied is a 2D plasma, which is bounded in one direction and unbounded or periodic in the other one. In this case one can apply a discrete Fourier transform along the periodic direction

$$A_{ij} = \frac{1}{2\pi} \sum_{k=0}^{Ny-1} A_i^k \mathrm{e}^{-\mathrm{i}2\pi j/N_y k} \tag{6.74}$$

with $A = \varphi, \rho$. By substituting this expression into (6.70) we obtain

$$\varphi_{i+1}^k - 2\left(1 + 2\left(\frac{\Delta x}{\Delta y}\sin\left(\frac{\pi k}{N_y}\right)\right)^2\right)\varphi_i^k + \varphi_{i-1}^k = -\frac{\Delta x^2}{\varepsilon_0}\rho_i^k \,. \tag{6.75}$$

It is easy to see that (6.75) is similar to the one for the 1D model considered above and can be solved in the same way. The main difference are the boundary conditions along the x-axis. E.g., if the plasma is bounded between two conducting walls, then $\varphi_0^k = \varphi_{N_g}^k = 0$ if $k > 0$, and for the $k = 0$-component we have exactly the same equation as for 1D with the same boundary condition.

6.5.3 Electromagnetic Case

For sufficiently strong fields and/or very fast processes it is necessary to solve the complete set of Maxwell's equations (6.3). It is obvious that corresponding solvers are more complicated than ones considered above. Correspondingly a detailed description of them is out of the scope of this work. Here we present just one of possible schemes, which is implemented in the XOOPIC code [8].

In order to ensure high speed and accuracy it is convenient to introduce a leap-frog scheme also for the fields. The leap-frog scheme is applied to the space coordinates too, which means that electric and magnetic fields are shifted in time by $\Delta t/2$, and different components of them are shifted in space by $\Delta x/2$. In other words:

(i) $\boldsymbol{E}$ is defined at $t = n\Delta t$ and $\boldsymbol{B}$ and $\boldsymbol{J}$ at $t = (n + 1/2)\Delta t$ time moments.
(ii) "i" components of the electric field and current density are defined at the points $x_i + \Delta_i/2$, x_k and x_j, and same component of the magnetic field at x_i, $x_k + \Delta_k/2$ and $x_j + \Delta_j/2$. Here x_s and Δ_s for $s = i, k, j$ denote the grid point and grid size along the s-axis. i, k and j denote the indices of the right-handed Cartesian coordinate system.

As a result the finite-differenced Ampere's and Faraday's laws in Cartesian coordinates can be written as

$$\frac{D^{i,t}_{i+1/2,k,j} - D^{i,t-\Delta t}_{i+1/2,k,j}}{\Delta t} = \frac{H^{j,t-\Delta t/2}_{i+1/2,k+1/2,j} - H^{j,t-\Delta t/2}_{i+1/2,k-1/2,j}}{\Delta x_k} - \frac{H^{k,t-\Delta t/2}_{i+1/2,k,j+1/2} - H^{k,t-\Delta t/2}_{i+1/2,k,j-1/2}}{\Delta x_j} - J^{i,t-\Delta t/2}_{i+1/2,k,j} \,, \tag{6.76}$$

$$\frac{B^{i,t+\Delta t/2}_{i,k+1/2,j+1/2} - B^{i,t-\Delta t/2}_{i,k+1/2,j+1/2}}{\Delta t} = \frac{D^{k,t}_{i,k+1/2,j+1} - D^{k,t}_{i,k+1/2,j}}{\Delta x_j} - \frac{D^{j,t}_{i,k+1,j+1/2} - D^{j,t}_{i,k,j+1/2}}{\Delta x_k} \,. \tag{6.77}$$

The solver works in the following way. The equations $\nabla \boldsymbol{D} = \rho$ and $\nabla \boldsymbol{B} = 0$ prescribe the initial electromagnetic fields. They remain satisfied due to Ampere's and Faraday's law, which are solved from the finite difference equations (6.76) and (6.77). The corresponding boundary conditions strongly depend on the simulated plasma model. E.g., at the wall representing an ideal conductor $E_\parallel = B_\perp = 0$, where $\parallel$ and $\perp$ denote the components parallel and normal to the wall, respectively.

As one can see, the components of the electromagnetic field obtained from (6.76) and (6.77) are defined at different time moments and spatial points than for the particle mover. Moreover, the current density obtained from the particle position is not defined at $t = (n + 1/2)\,\Delta t$ as it is required for Ampere's law (6.76). Hence, it is necessary to additionally couple the particle and field solvers [3].

It is useful to derive a general stability criteria for the electromagnetic case. For this we consider electromagnetic waves in vacuum

$$\boldsymbol{A} = \boldsymbol{A}_0 \mathrm{e}^{\mathrm{i}\boldsymbol{k}\boldsymbol{x} - \omega t} \,, \tag{6.78}$$

with $\boldsymbol{A} = \boldsymbol{E}, \boldsymbol{B}$. After substitution of (6.78) into field equations (6.76) and (6.77) and trivial transformations we obtain

$$\left(\frac{\sin(\omega t/2)}{c\Delta t}\right)^2 = \sum_{i=1}^{3} \left(\frac{\sin(k_i x_i/2)}{\Delta x_i}\right)^2 \,, \tag{6.79}$$

where $c = \sqrt{1/\varepsilon_0 \mu_0}$ is the speed of light. It is obvious that the solution is stable (i.e. $\mathrm{Im}\omega < 0$) if

$$(c\Delta t)^2 < \left(\sum_{i=1}^{3} \frac{1}{\Delta x_i^2}\right)^{-1} \,. \tag{6.80}$$

Often, this so called Courant condition requires unnecessary small time step for the particle mover. In order to relax it one can introduce separate time steps for field and particles. This procedure is called "sub-cycling" [3].

The routines described above namely: The field solver, the particle mover with proper boundary conditions and the particle source, weighting of particles and fields represent a complete PIC code in its classical understanding. Starting from 1970s a number of PIC codes include different models of particle collisions. Today the majority of PIC codes include at least some kind of collision operator, which have to be attributed to a PIC technique. These operators are usually based on statistical methods and correspondingly are called Monte Carlo (MC) models. Often different authors use the name PIC-MC code. The MC simulations represent an independent branch in numerical physics and the interested reader can find more on MC method in corresponding literature (e.g., see Part II). Below we consider the main features of the MC models used in PIC codes.

6.6 Particle Collisions

6.6.1 Coulomb Collisions

The forces acting on the particles in a classical PIC scheme correspond to macro fields, so that the simulated plasma is assumed to be collisionless. In order to simulate a collisional plasma it is necessary to implement corresponding routines. Moreover, the field solver is organized in such a way that self-forces are excluded, hence, the field generated by a particle inside the grid cell decreases with decreasing distance from this particle. As a result, inter-particle forces inside grid cells are underestimated (see Fig. 6.5). Hence, they can be (at least partially) compensated by introducing the Coulomb collision operator.

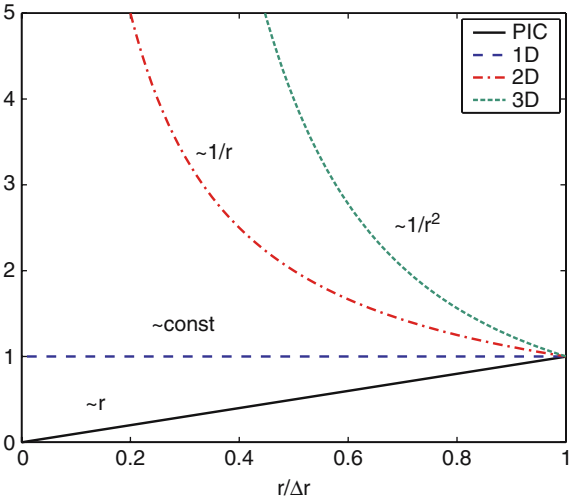

Fig. 6.5. Inter-particle forces inside grid cell

The first codes simulating Coulomb collisions were the particle-particle codes simulating the exact interaction between each particle pair. Of course this method, which scales as N^2 can not be used in contemporary PIC simulations. Later different MC models have been developed.

The simplest linear model assumes that the particle distribution is near to a Maxwellian and calculates an average force acting on particles due to collisions [16]. Although this is the fastest operator it probably can not be used for most of kinetic plasma simulations, when particle distributions are far from the Maxwellian. A nonlinear analogue of this model has been introduced in [17]. Here, the exact collision inter-particle inter-particle is obtained from the particle velocity distribution function. Unfortunately, the number of particles required for building up a sufficiently accurate velocity distribution is extremely large (see [18]), which makes it practically impossible to simulate large systems.

Most of nonlinear Coulomb collision operators used in our day PIC codes are based on the binary collision model introduced in [19]. In this model each particle inside a cell is collided with one particle from the same cell. This collision operator conserves energy and momentum and it is sufficiently accurate. The main idea is based on the fact that there is no need to consider Coulomb interaction between two particles separated by a distance larger than the Debye radius λ_D (e.g., see [20]). Since a typical size of the PIC cell is of the order of λ_D, the interaction between the particles in different cells can be neglected. This method consists of the following three steps (see Fig. 6.6):

(i) First, all particles are grouped according to the cells where they are located;
(ii) Then these particles are paired in a random way, so that one particle has only one partner;
(iii) Finally, the paired particles are (statistically) collided.

The latter is not trivial and we consider it in some detail.

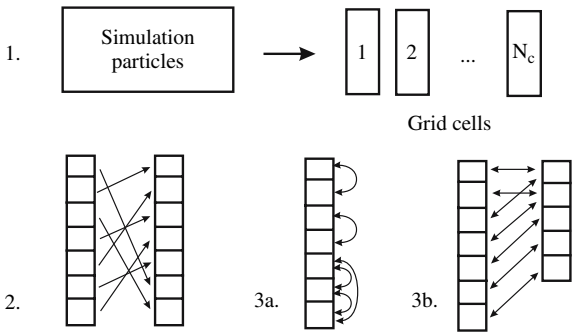

Fig. 6.6. Binary collision model from [19]. **1**. Grouping of particles in the cells; **2**. Randomly changing the particle order inside the cells; **3a**. Colliding particles of the same type; **3b**. Colliding particles of different types

According to the momentum and energy conservation constrains, we can express the after-collision velocities of particles $\boldsymbol{V}_1'$, $\boldsymbol{V}_2'$ via their before-collision values $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ [21]

$$\boldsymbol{V}_1' = \boldsymbol{V}_1 + \frac{m_2}{m_1+m_2}\Delta\boldsymbol{V} \quad \text{and} \quad \boldsymbol{V}_2' = \boldsymbol{V}_2 - \frac{m_1}{m_1+m_2}\Delta\boldsymbol{V} \tag{6.81}$$

with $\Delta\boldsymbol{V} = \boldsymbol{V}' - \boldsymbol{V}$, $\boldsymbol{V} = \boldsymbol{V}_2 - \boldsymbol{V}_1$, $\boldsymbol{V}' = \boldsymbol{V}_2' - \boldsymbol{V}_1'$ and $V'^2 = V^2$. As we see the calculation can be reduced to the scattering of the relative velocity $\boldsymbol{V}$

$$\Delta\boldsymbol{V} = \left(\widehat{O}\left(\chi,\psi\right) - 1\right)\boldsymbol{V}\ , \tag{6.82}$$

where $\widehat{O}\left(\alpha,\beta\right)$ is the matrix corresponding to the rotation on angles α and β (see [19]). χ and ψ represent the scattering and azimuthal angles.

The scattering angle χ is calculated from a corresponding statistical distribution. By using the Fokker-Plank collision operator one can show (see [22]) that during the time Δt_c the scattering angle has the following Gaussian distribution

$$\begin{aligned} P\left(\chi\right) &= \frac{\chi}{\left\langle\chi^2\right\rangle_{\Delta t_c}} \mathrm{e}^{-\chi^2/\left(2\left\langle\chi^2\right\rangle_{\Delta t_c}\right)}\ , \\ \left\langle\chi^2\right\rangle_{\Delta t_c} &\equiv \frac{e_1^2 e_2^2}{2\pi\varepsilon_0^2}\,\frac{n\Delta t_c \Lambda}{\mu^2 V^3}\ . \end{aligned} \tag{6.83}$$

Here $e_{1,2}$ and $\mu = m_1 m_2/\left(m_1{+}m_2\right)$ denote the charge and reduced mass of the collided particles, respectively. n and Λ are the density and the Landau logarithm [20], respectively. The distribution (6.83) can be inverted to get

$$\chi = \sqrt{-2\left\langle\chi^2\right\rangle_t \ln R_1}\ . \tag{6.84}$$

Correspondingly, the azimuthal angle ψ is chosen randomly between 0 and 2π

$$\psi = 2\pi R_2\ . \tag{6.85}$$

R_1 and R_2 are random numbers between 0 and 1.

Finally, the routine for two-particle collision is reduced to the calculation of expressions (6.81), (6.82), (6.84), and (6.85).

The Coulomb interaction is a long range interaction, when a cumulative effect of many light collisions with small scattering angle represents the main contribution to the collisionality. Accordingly, the time step for the Coulomb collisions Δt_c should be sufficiently small: $\left\langle\chi^2\right\rangle_{\Delta t}\left(V = V_T\right) \ll 1$. It is more convenient to formulate this condition in the equivalent following form

$$\nu_{\mathrm{c}}\Delta t_{\mathrm{c}} \ll 1 \qquad \text{and} \qquad \nu_{\mathrm{c}} = \frac{e_1^2 e_2^2}{2\pi\varepsilon_0^2}\,\frac{n\Lambda}{\mu^2 V_{\mathrm{T}}^3}\ , \tag{6.86}$$

where ν_{c} is the characteristic relaxation time for the given Coulomb collisions [23] and V_{T} is the thermal velocity of the fastest collided particle species. Although usually $\Delta t_{\mathrm{c}} \gg \Delta t$, the binary collision operator is the most time consuming part of the PIC code. Recently, in order to speed up the collisional plasma simulations a number of updated versions of this operator have been developed (e.g., see [6, 24] and [25]).

6.6.2 Charged-Neutral Particle Collisions

Under realistic conditions the plasma contains different neutral particles, which suffer collisions with the plasma particles. The corresponding collision models used in PIC codes can be divided in two different schemes: Direct Monte-Carlo and null-collision models.

The direct Monte-Carlo model is a common MC scheme when all particles carry information about their collision probability. In this scheme all particles have to be analyzed for a collision probability. Hence, the direct MC requires some additional memory storage and sufficiently large amount of the CPU time.

The null collision method (see [26] and [27]) requires a smaller number of particles to be sampled and it is relatively faster. It uses the fact that in each simulation time step only a small fraction of charged particles suffer collisions with the neutrals. Hence, there is no necessity to analyze all particles. As a first step the maximum collision probability is calculated for each charged particle species

$$P_{\max} = \left(1 - \mathrm{e}^{-\sigma n \Delta s}\right)_{\max} = 1 - \mathrm{e}^{-(\sigma V)_{\max} n_{\max} \Delta t} \,, \tag{6.87}$$

where $\sigma = \sum \sigma_i (V)$ and n are the total collision cross-section, i.e. the sum of cross-sections σ_i for all possible collision types and the neutral density, respectively. $\Delta s = V \Delta t$ is the distance, which the particle travels per Δt time. Accordingly, the maximum number of particles which can suffer a collision per Δt time is given as $N_{nc} = P_{\max} N \ll N$. As a result only N_{nc} particle per time step have to be analyzed. These N_{nc} particles are randomly chosen, e.g., by using the expression $i = R_j N$ with $j = 1, \ldots, N_{nc}$, where i is the index of the particle to be sampled and R_j are the random numbers between zero and one. The sampling procedure itself includes the calculation of the collision probability of a sampled particle and choosing which kind of collision it should suffer (if any). For this a random number R is compared to the corresponding relative collision probabilities: if

$$R \leq \frac{P_1}{P_{\max}} = \frac{1 - \mathrm{e}^{-\sigma_1 V n \Delta t}}{P_{\max}} \approx \frac{n \sigma_1 (V) V}{(\sigma V)_{\max} n_{\max}} \,, \tag{6.88}$$

a type one collision takes place; else if

$$R \leq \frac{P_1 + P_2}{P_{\max}} \approx \frac{n V \left(\sigma_1 (V) + \sigma_2 (V)\right)}{(\sigma V)_{\max} n_{\max}} \,, \tag{6.89}$$

a type two collision takes place, and so on. If

$$R > \frac{\sum P_i}{P_{\max}} \approx \frac{n V \sum \sigma_i (V)}{(\sigma V)_{\max} n_{\max}} \tag{6.90}$$

no collision takes place.

The difference between the nonlinear and linear null collision methods is the way how the collided neutral particle is treated. In the linear models the neutral

velocity is picked up from the prescribed distribution (usually the Maxwellian distribution with the given density and temperature profiles). Contrary to this, in the nonlinear case the motion of neutral particles is resolved in the simulation, and the collided ones are randomly chosen from the same cells, where the colliding charged-particle are.

When the collision partners and corresponding collision types are chosen, the collision itself takes place. Each collision type needs a separate consideration, so that here we discuss the general principle.

The easiest collisions are the ion-neutral charge-exchange collisions. In this case the collision is reduced to an exchange of velocities

$$\boldsymbol{V}_1' = \boldsymbol{V}_2 \qquad \text{and} \qquad \boldsymbol{V}_2' = \boldsymbol{V}_1 \,. \tag{6.91}$$

The recombination collisions are also easy to implement. In this case the collided particles are removed from the simulation and the newly born particle, i.e. the recombination product, has the velocity derived from the momentum conservation

$$\boldsymbol{V}_{\text{new}} = \frac{m_1 \boldsymbol{V}_1 + m_2 \boldsymbol{V}_2}{m_{\text{new}}} \,. \tag{6.92}$$

The elastic collisions are treated in a similar way as the Coulomb collisions using (6.81). The scattering angle depends on the given atomic data. E.g., often it is assumed that the scattering is isotropic

$$\cos\chi = 1 - 2R \,. \tag{6.93}$$

In order to save computational time during the electron-neutral elastic collisions the neutrals are assumed to be at rest. Accordingly, in spite of resolving (6.81) a simplified expression is used for the calculation of the after-collision electron velocity

$$V_{\text{e}}' \approx V_{\text{e}} \sqrt{1 - \frac{2m_{\text{e}}}{M_{\text{n}}} \left(1 - \cos\chi\right)} \,. \tag{6.94}$$

Excitation collisions are done in a similar way as the elastic ones, just before the scattering the threshold energy E_{th} is subtracted from the charged particle energy

$$\boldsymbol{V} \;\Rightarrow\; \boldsymbol{V}' = \boldsymbol{V} \sqrt{1 - \frac{E_{\text{th}}}{E}} \;\Rightarrow\; \text{scattering} \;\Rightarrow\; \boldsymbol{V}'' \,. \tag{6.95}$$

Important to note is that one has to take care on the proper coordinate system, e.g., in (6.95) the first transform should be done in a reference system, where the collided neutral is at rest.

Implementation of inelastic collisions when secondary particles are produced is case dependent. E.g., in electron-neutral ionization collisions, first the neutral particle is removed from the simulation and a secondary electron-ion pair is born. The velocity of this ion is equal to the neutral particle velocity. The velocity of electrons is calculated in the following way. First, the ionization energy is subtracted

from the primary electron energy and then the rest is divided between the primary and secondary electrons. This division is done according to given atomic data. After finding these energies the electrons are scattered on the angles χ_{prim} and χ_{sec}.

In a similar way the neutral-neutral and inelastic charged-charged particle collisions can be treated.

6.7 Final Remarks

The material presented above represents just the basics of PIC codes. Nowadays PIC codes use different optimizations including paralleling of the code and memory optimizations (see [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]), developing of more efficient collision operators (see [6, 24, 25, 29]) and grid-less solvers (see [30] and [31]).

It has to be noted, that in the present work we do not consider the analytic theory of the PIC simulation, which can provide useful information on possible numerical oscillation modes and can help to better understand number of conditions to be satisfied in PIC simulation. The interested reader can find the corresponding material in [3] and [4].

References

1. O. Buneman, Phys. Rev. **115**(3), 503 (1959) 161
2. J. Dawson, Phys. Fluids **5**(4), 445 (1962) 161
3. C. Birdsall, A. Langdon, *Plasma Physics Via Computer Simulation* (McGraw-Hill, New York, 1985) 161, 163, 164, 169, 170, 175, 176, 182, 188
4. R. Hockney, J. Eastwood, *Computer Simulation Using Particles* (IOP, Bristol and New York, 1989) 161, 163, 165, 172, 175, 176, 188
5. V. Decyk, Comput. Phys. Commun. **87**(1-2), 87 (1995) 161, 188
6. D. Tskhakaya, R. Schneider, J. of Comp. Phys. **225**(1), 829–839 (2007) 161, 185, 188
7. J. Verboncoeur, M. Alves, V. Vahedi, C. Birdsall, J. Comput. Phys. **104**(2), 321 (1993) 162, 168, 177, 1
8. J. Verboncoeur, A. Langdon, N. Gladd, Comput. Phys. Commun. **87**(1–2), 199 (1995) 162, 181, 188
9. D. Barnes, T. Kamimura, J.N. Le Boeuf, T. Tajima, J. Comput. Phys. **52**(3), 480 (1983) 166, 188
10. D. Tskhakaya, S. Kuhn, Contrib. Plasma Phys. **42**(2–4), 302 (2002) 168, 188
11. K. Cartwright, J. Verboncoeur, C. Birdsall, J. Comput. Phys. **162**(2), 483 (2000) 168, 188
12. D. Tskhakaya, S. Kuhn, Plasma Phys. Contr. F. **47**, A327 (2005) 168, 188
13. F. F. Collino, T. Fouquet, P. Joly, J. Comput. Phys. **211**(1), 9 (2006) 175, 188
14. V. Vahedi, G. DiPeso, J. Comput. Phys. **13**(1), 149 (1997) 177, 179, 180, 188
15. W. Press, S. Teukolsky, W. Vetterling, B. Flannery, *Numerical Recipes in C* (Cambridge University Press, Cambridge, New York Port Chester, Melbourne, Sydney, 2002) 179, 181, 188
16. A. Bergmann, Contrib. Plasma Phys. **38**, 231 (1998) 184, 188
17. O. Batishchev, X. Xu, J. Byers, R. Cohen, S. Krasheninnikov, T. Rognlien, D. Sigmar, Phys. Plasmas **3**(9), 3386 (1996) 184, 188
18. O. Batishchev, S. Krasheninnikov, P. Catto, A. Batishcheva, D. Sigmar, X. Xu, J. Byers, T. Rognlien, R. Cohen, M. Shoucri, I. Shkarofskii, Phys. Plasmas **4**(5), 1672 (1997) 184, 188
19. T. Takizuka, H. Abe, J. Comput. Phys. **25**(3), 205 (1977) 184, 185, 188

20. N. Krall, A. Trivelpiece, *Principles of Plasma Physics* (San Francisco Press, Inc., Box 6800, San Francisco, 1986) 184, 185, 188
21. L. Landau, E. Lifshitz, *Course of TheoreticalPhysics*, vol. 1, Mechanics (Pergamon Press, Oxford-London-Paris, 1960) 185, 188
22. R. Shanny, J. Dawson, J. Greene, Phys. Fluids **10**(6), 1281 (1967) 185, 188
23. D. Book, *NRL Plasma formulary* (Naval Research Laboratory, Washington D.C., 1978) 185, 188
24. K. Nanbu, Phys. Rev. E **55**(4), 4642 (1997) 185, 188
25. A. Bobylev, K. Nanbu, Phys. Rev. E **61**(4), 4576 (2000) 185, 188
26. C. Birdsall, IEEE T. Plasma Sci. **19**(2), 65 (1991) 186, 188
27. V. Vahedi, M. Surendra, Comput. Phys. Commun. **87**, 179 (1995) 186, 188
28. K. Bowers, J. Comput. Phys. **173**(2), 393 (2001) 188
29. K. Matyash, R. Schneider, A. Bergmann, W. Jacob, U. Fantz, P. Pecher, J. Nucl. Mater. **313-316**, 434 (2003) 188
30. A. Christlieb, R. Krasny, J. Verboncoeur, IEEE T. Plasma Sci. **32**(2), 384 (2004) 188
31. A. Christlieb, R. Krasny, J. Verboncoeur, Comput. Phys. Commun. **164**(1–3), 306 (2004) 188