Particle-In-Cell Model

This document describes the model for Monte Carlo simulation of charged particles as implemented in PICVolDistrib class. It both includes the charge deposit and particle trajectories integration.

Charge deposit

Charge density is computed form the sampled particle distribution (Monte Carlo) following Particle-In-Cell scheme (PIC). When density is requested by the code, each particle deposits its density on the neighbouring nodes, with a linear weighing. For the current 3D unstructured volume mesh, it amounts to depositing charge on the four nodes of the tetrahedron containing the particle proportionally to its barycentric coordinates.

Trajectory integration

Basic model

The *basic model* for particle trajectory integration is a leapfrog scheme. Particle position and acceleration are computed at time $t^{(n)}$ with spacing dt, while velocities are computed at intermediate times $t^{(n+1/2)} = t^{(n)} + dt/2$. Although this scheme is coded as first order (v -> v + $a.dt$, x -> x + $v.dt$) the shifting of velocity to $t^{(n+1/2)}$ makes it accurate to second order in dt (error in $O(dt^3)$). This however constraints dt to be constant for a particle, hence for all particles of a population in practice. This is why user control for population time step is very important (or at least in some situations, see next paragraph). The user must check manually that the time step the code determines from user inputs (see Controlling Num from UI) is correct to ensure particle steps significantly smaller than cell size (time steps dt for each populations are displayed on screen during execution).

Improvement in 2005

A significant improvement with respect to this basic model was implemented in 2005. In this *advanced model*, it is taken advantage of the specificities of current SPIS electric field model to perform an exact integration of particle trajectories. Since the potential is currently considered as step-wise linear (in each tetrahedron), the electric field is constant in each cell, which yields a parabolic trajectory in each cell. When possible the particle trajectories are thus integrated exactly, following a new parabolic arc in each tetrahedron. The limitations to this possibility are the following:

- presence of a magnetic field
- special shape of potential (non linear) in the vicinity of thin wires (1D) or thin plates (2D)

in which case the code automatically switches back to the basic leapfrog model (for the whole domain, not simply the local tetrahedra around wires and plates which do not have linear potential).

NB1: in presence of a magnetic field, the trajectories are still analytical (accelerated screw) only the intersections with the tetrahedra planes are not. It could thus be possible to also perform an analytical integration provided some typically dichotomy-like method is implemented for intersections. So, in future this solver could be improved with analytical

integration all over the mesh even in presence of *B* field, with another method in the specific tatrahedra close to wires and plates (leapfrog or better).

NB2: Of course, using the specificities of the field model makes the code less modular. The basic leapfrog model can work with whatever field model, provided it can supply electric field at particle positions, for example an analytical *E* field, or even an *E* field computed an a different mesh, etc.

Improvements in SPIS v4

Major improvements were brought to particle integration in SPIS v4 (2007-09).

First the switch between the exact integration method and the iterative method was made local. When crossing tetrahedra with a constant electric field and no B field, the exact integration method is used. In tetrehedra with non uniform forces (either a non uniform electric field or a B field) the integration is performed through an iterative method.

Second, the iterative method was improved from leapfrog (2^{nd} order method) into a Runge-Kutta Cash-Karp method, which is 4^{th} order exact with a control of the accuracy. This allowed implementing a local subcycling to control the accuracy for each particle.

Finally, on the software point of view, the particle pusher that implements these algorithms (the solver of particle movement equations) was separated from the PIV volume distribution. As a consequence a single PICVolDistrib class can incorporate variable pushers. As of SPIS v4.0 two pushers exist (implement the interface):

- the BasicPusher implements the old method (global switch between exact and leapfrog)
- while the ComplexPusher implements the new one (local switch between exact and RKCK)