GRAPHICAL FRONT-END AND OBJECT-ORIENTED DESIGN FOR IONEX, AN ION EXTRACTION MODELING CODE*

L. Grubert, N. Barov, B. Cluggish, S. Galkin, J.S. Kim, FAR-TECH, Inc., San Diego, CA, U.S.A.

Abstract

IonEx is a new hybrid, meshless, cross-platform, 2D code which can model the extraction of ions from a plasma device. The application includes a user-friendly Graphical User Interface (GUI), which will contain a geometry editor for specifying the domain. The design of IonEx utilizes the object-oriented functionality of C++, which provides an efficient means of incorporating magnetic fields, an arbitrary geometry, and multiple ion species into a simulation. Visualization of the resulting trajectories and emittances is accomplished through the GUI; openGL is used to accelerate the graphics. In this paper we will briefly review the physics and computational methods used, highlight important aspects of the object-oriented design, discuss the primary features of the GUI, describe the current status of IonEx, and present some simulation results.

ION EXTRACTION MODELING

The IonEx code was designed to model the extraction of ions from the plasma in an Electron Cyclotron Resonance Ion Source (ECRIS). This section provides a brief review of the physical model and computational methods used; for more details see [1,4].

Physical Model

IonEx uses a hybrid model which describes the ions as particles and the electrons as a massless Boltzmann fluid. The magnetic field is given; the electrostatic field is solved for self-consistently. Each iteration includes solving for the electrostatic field, a particle-pushing step, and the re-distribution of charge from the particles to the computational points. For the particle-pushing step, we treat the ions as macroparticles and solve the equations of motion:

$$d\mathbf{r}_s / dt = \mathbf{v}_s$$

$$dm_s\mathbf{v}_s / dt = q_s(\mathbf{E} + \mathbf{v}_s \times \mathbf{B})$$

where $\mathbf{r}_s$, $\mathbf{v}_s$, $\mathbf{m}_s$, and $\mathbf{q}_s$ denote, respectively, the r-coordinate, velocity, mass, and charge of the *s*-th macroparticle, and **E** and **B** give the value of the electrostatic and magnetic fields at the particle's location. To solve for the electrostatic field, Poisson's equation (with Boltzmann electron term) is used, where ρ_i and ρ_e are the ion and electron charge densities; ρ_0 and Φ_0 are the ion charge density and potential in the plasma; and ε_0 is the permittivity of free space:

$$\Delta\phi = -\frac{\rho_i + \rho_e}{\varepsilon_0} = -\frac{\rho_i - \rho_0 \exp(e(\phi - \phi_0)/T_e)}{\varepsilon_0}$$

*Work supported by US DOE SBIR program (Nuclear Physics Division)

Computational Methods Used

The IonEx code uses the Particle-in-Cloud-of-Points (PICOP) Method [1]. The PICOP method is similar to the Particle-in-Cell (PIC) Method, but the domain is represented by a cloud of points instead of a mesh. It is an adaptive meshless method, with point adaptation taking place after a specified number of particle-pushing iterations. The IonEx code includes a point generator which is responsible for creating and adapting points according to a monitor function based on the current solution.

Within the PICOP framework, standard numerical methods are utilized for solving the sparse matrix equation and the Ordinary Differential Equation (ODE). Both Gauss-Seidel and Newton's method are used to solve Poisson's equation. The equations of particle motion are computed using a fourth-order Runge-Kutta scheme with an adaptive time step. These methods are programmed into the code using object-oriented design techniques.

OBJECT-ORIENTED DESIGN

The programming behind IonEx is guided by the principles of object-oriented design [5]. Objects are implemented as classes in C++. Encapsulation is used to hide the details of functions within those classes. This greatly increases the code's re-usability and makes it easy to maintain. Classes support both the GUI and the computational code. Here, we will discuss only the classes used for the simulation part of the code.

The primary objects involved in the simulation are the field solver, point generator, particles, and IonEx driver classes. The driver is the interface between the computational code and any other object (such as the GUI) which needs to run an ion extraction simulation. The particles class is responsible for the particle-pushing part of the code. The point generator creates and adapts the points. The field solver handles the set-up and solving of Poisson's equation.

These primary classes utilize several supporting classes. We will discuss a few of these with a bit more detail: the ion species, magnetic field, and ODE solver classes (supporting the particles class); and the geometry and boundary condition classes (primarily supporting the field solver and point generator).

The ion species class contains information specific to each species, such as its atomic mass, charge state, and initial energy and velocity in the z-direction. Additionally, through the GUI, a user can assign a name and color to each species, for easy identification. The

particles class stores an array of instances of the ion species class, one for each species. We can then simply loop through the array to handle any calculations which are specific to one species.

The magnetic field class is responsible for providing the magnetic field components at any particular position in the domain. The B-field can be stored in a variety of different formats within this class: described by a polynomial, specified along the z-axis, or prescribed at points throughout the domain.

The ODE solver is a far more complicated object than either the magnetic field or ion species class. This is because at every time step of the solve, communication with several other classes is required. To solve the equations of motion for the macroparticles, the ODE solver needs to have access to the electrostatic field, the magnetic field, the domain geometry, and various functions within other classes. The solution implemented in IonEx is to have a base ODE solver class. By inheriting from this base class, we can pass arbitrary data to the ODE solver and implement the problem-specific calculations required within the solver. This also gives us the ability to easily replace the core ODE-solving method with another one.

The IonEx code is able to run a simulation with an arbitrary geometry, by the generalized geometry and boundary condition classes. The geometry class contains a description of the boundary of the domain, including information required to re-construct curves. It handles such operations as computing the domain volume, correctly adding points on the boundary (during refinement), and determining whether a given point is inside or outside the domain. The base boundary condition class is used to derive the Dirichlet and Neumann classes. These classes assist with estimating the coefficients of the Laplacian matrix and setting up the right-hand side for the Poisson solve.

GRAPHICAL USER INTERFACE

IonEx has a Graphical User Interface which allows the user to specify the domain, setup the simulation parameters, run the simulation, and visualize the results. The GUI was created using Qt, which is a C++ application development framework for creating cross-platform GUI applications [2].

Currently, the domain is specified through a text input file. This will be replaced by an interactive geometry editor which allows the user to construct the domain boundary in a manner similar to a Computer-Aided Design (CAD) program. Drawing tools will be available for a canvas, and the ability to graphically assign boundary conditions to the boundary elements will be provided. An expression evaluator will also be included, which allows the user to create and evaluate numeric expressions.

The GUI also provides a user-friendly means of setting up the simulation parameters. The parameters can either be imported from a text file or entered through dialogs. The values given for the input parameters are displayed in a tree on the left panel. Interaction with the tree allows the user to edit or delete particular items in the tree.

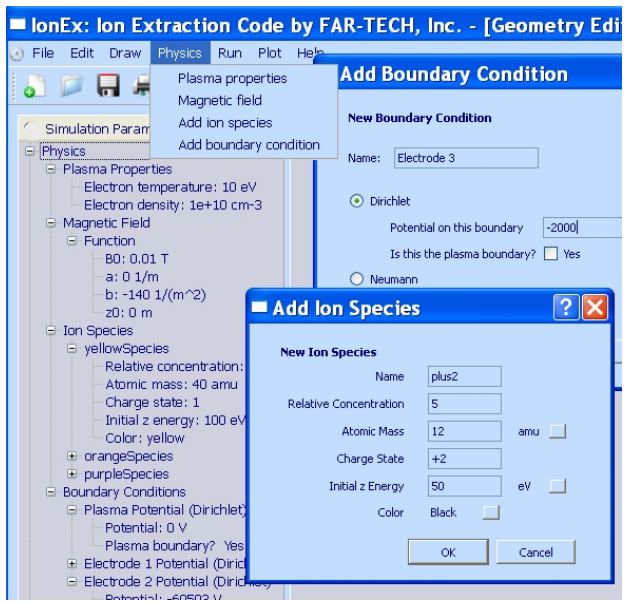

Figure 1: IonEx Screenshot.

Once the simulation parameters and boundary geometry have been specified, the simulation can be run from the GUI. It runs in a separate thread to keep the GUI active (since the code is computationally intensive). As the simulation runs, trajectory plots and status updates are displayed; this communication between the computational code and the GUI is accomplished through the use of Qt's signal/slot mechanisms. The simulation can be aborted at any time. An output file containing particle positions, velocities, etc. is generated when the simulation ends.

After the simulation ends (or is aborted), the user can view multiple plots of the domain (computational points and boundary conditions), emittance (at a specified z-value), and particle trajectories (as an overlay on a contour plot of the electrostatic field). Right-clicking on a plot brings up menu options which allow the user to change the ion species or the z-value for that plot. The plots are individual windows within a Multiple Document Interface (MDI) area, which is the central widget for the application. This gives the user complete control over the number, size, and arrangement of plots. The graphics used in the plots are the result of combining openGL [3] and the Qt Painter class. OpenGL handles the more computationally expensive operations, such as generating the contour plot, while Painter is used to add axes labels.

STATUS OF IONEX

Many improvements have been made to IonEx in recent months. The entire computational code has been successfully ported to c++ (in preparation for distribution). A Graphical User Interface has been added, which includes the ability to enter simulation parameters,

run the simulation, and display the results. The code now includes the effects of the magnetic field and the ability to run multiple-species simulations. The point generator has been incorporated, and a more general geometry can now be used. The code has been compiled and tested under both Windows and Linux.

Some additional work remains for this project. The geometry editor is not yet included in the application, and more GUI features need to be added to give the user more control during the post-processing phase. Future plans include linking IonEx with the rest of the suite of ECR Ion Source simulation codes that FAR-TECH is developing: MCBC (Monte-Carlo Beam Capture) and GEM (Generalized ECRIS plasma Modeling). IonEx will also be ported to the Mac. The 3D version of IonEx will be completed.

SIMULATION RESULTS

This section presents the results obtained by running an ion extraction simulation using He+1 and He+2 with equal current and a nearly uniform magnetic field. The input parameters are given below.

Table 1: Input Parameters for IonEx

Input Parameter	Value
Electron Density	2e16 m-3
Electron Temperature	100 eV
Initial Ion Energy	100 eV
Species	He+1 and He+2
Number of particles tracked	200 of each species
Maximum Magnetic Field	1 T
Extraction Voltage	60 kV

Figure 2 shows the emittance plots for each species. In Figure 3, the trajectories (with both species together) are displayed. The potential is also plotted behind the trajectories.

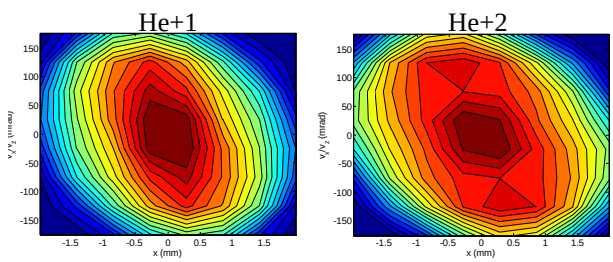

Figure 2: Emittance Plots.

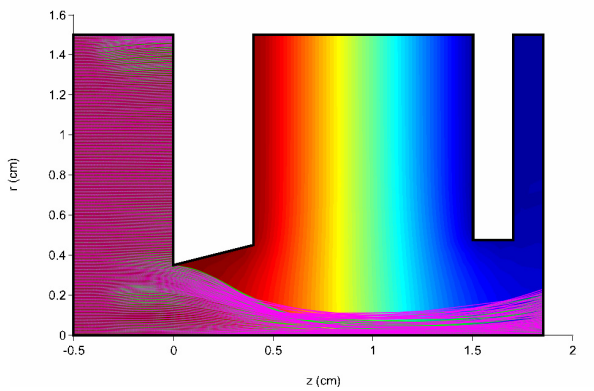

Figure 3: Trajectory and Equipotential Plot.

Table 2 displays the extracted beam parameters at the extraction end of the domain (at z = 1.85cm). The root mean square of the radius (R), energy (E), and emittance is given for each species.

Table 2: Extracted Beam Parameters

	He+1	He+2
R_{rms}(mm)	1.29	1.48
E_{rms}(keV) [Total]	57.2	114
E_{rms}(keV) [Transverse]	1.21	2.51
2D emittance (micron)	19.9	24.7

REFERENCES

[1] S. Galkin, B. Cluggish, J.S. Kim, and S. Medvedev, "Advanced PICOP Algorithm with Adaptive Meshless Field Solver," Proc. Pulsed Power and Plasma Science Conference, 2007.

[2] J. Blanchette and M. Summerfield, "C++ GUI Programming with Qt 4," Prentice Hall, 2008.

[3] M. Woo, J. Neider, T. Davis, and D. Shreiner, "OpenGL Programming Guide," Addison Wesley, 1999.

[4] B. Cluggish, S. Galkin, J.S. Kim, "Modeling Ion Extraction from an ECR Ion Source," Proc. Particle Accelerator Conference, 2007.

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley Professional Computing Series, 1995.