OOPIC Pro™
Object-Oriented Particle-in-Cell Simulation

User's Manual

Version 1.0

Tech-X Corporation
5621 Arapahoe Avenue, Suite A
Boulder, CO 80305
http://www.txcorp.com

August 11, 2004

1 Preface

This manual documents OOPIC Pro, version 1.0.

OOPIC Pro uses the same physics kernel as XOOPIC. OOPIC Pro has a newer graphical user interface (GUI), which can run on Microsoft Windows and all Linux platforms, while XOOPIC has an X11-based GUI. We use the term OOPIC to refer to the common physics kernel.

The original XOOPIC (X11-based Object Oriented Particle in Cell) simulation code was developed at the University of California at Berkeley, beginning in 1995, by members of the Plasma Theory and Simulation Group (PTSG). Since 1998, Tech-X Corporation has been working in collaboration with PTSG staff to improve and generalize the OOPIC physics kernel. The new GUI for OOPIC Pro was developed by Tech-X Corporation.

1.1 Copyright Info

2 Introduction

OOPIC Pro is a 2D particle-in-cell (PIC) code, with electrostatic and electromagnetic field solvers, and support for x-y (slab) and r-z (cylindrical) geometries. OOPIC Pro simulates physical systems that can include plasmas (a plasma is a hot, charged gas), beams of charged particles, externally generated electric and magnetic fields, low-to-moderate density neutral gasses, and a wide variety of boundary conditions.

The OOPIC Pro physics kernel has been used by researchers around the world since 1995 to simulate a wide range of challenging problems, such as plasma display panels, ion implantation, high-power microwave devices, and next-generation particle accelerator concepts. The code is somewhat unique among PIC codes in its ability to handle ionization of background neutral gasses via electron impact or field-induced tunneling effects. More detailed information is provided by Ref. 1 and 2 in Section 8 below.

OOPIC Pro is an object-oriented code, written in C++, and thus can be altered and extended in a straightforward fashion by experienced C++ developers. For those users interested in modifying or adding to the existing code, please contact Tech-X Corporation at sales@txcorp.com or call (303) 444-2416 for information.

OOPIC Pro provides a convenient and intuitive GUI for use on Microsoft Windows, Mac OS X, and Linux systems, as well as a batch mode to run jobs from the command line.

2.1 Objective of This Manual

This manual documents the simulation code OOPIC Pro, Version 1.0, which includes the OOPIC physics kernel, version 3.0 beta. This version of OOPIC Pro is built on the QScimpl scientific graphics package, version 1.0 beta 3. For a discussion of QScimpl, see URL http://www.txcorp.com/products/QScimpl/.

It is assumed that the reader has no prior experience using XOOPIC or OOPIC Pro, but that there is some level of familiarity with particle-in-cell (PIC) methods and theory. This manual will enable the user to

- create OOPIC input files that specify a wide variety of physical configurations,

- effectively execute the OOPIC Pro application to simulate the specified systems, and
- use the interactive visual diagnostics of OOPIC Pro to view and understand the results.

2.2 Organization of This Manual

This manual is divided into several sections. The next section, Section 3, is for new users. It includes installation instructions and a brief overview of the input file structure and parameter groups.

- Section 4 describes the OOPIC Pro graphical user interface (GUI) in detail. This includes instruction on how to start and run the code. The user is instructed on how to create and manipulate diagnostic plots and how to produce output dump files of those plots and of the entire simulation. There is also information about how to create and view movies of the plots.
- Section 5 describes the syntax, the special features and the three block structures of OOPIC input files in detail.
- Section 6 describes the many parameter groups that are used to specify the simulated system. A look through this section will quickly reveal the capabilities and limitations of the OOPIC physics kernel.
- Section 7 provides a number of examples of OOPIC Pro usage to model different types of devices with different geometries and inputs.
- Section 8 is a list of references that the user can refer to for detailed information about the methods employed in OOPIC Pro.
- Appendix 9 is an appendix describing the command-line options that one can use. This section also describes how to run OOPIC Pro in parallel.
- Appendix 10 is an appendix describing the cross-platform binary dump file format used by OOPIC Pro. These dump files are used primarily to checkpoint long runs and to preserve the final state of a simulation to be examined at a later time.

Contents

3 Getting Started

OOPIC Pro comes with an intuitive GUI that aids the user in understanding the results of the specified PIC simulation. However, correct and effective use of OOPIC Pro requires that the user develop a valid input file, which must accurately specify the physical system to be simulated. This section describes how to install OOPIC Pro and provides an overview of the structure of input files. The GUI is described in Section 4 below.

3.1 Installation

Here the procedure for installing OOPIC Pro on various platforms is described.

3.1.1 Evaluation Version of OOPIC Pro

An evaluation version of OOPIC Pro is available for download from http://www.txcorp.com/products/oopicpro/. The evaluation version is fully functional, but will expire in 30 days. Information on purchasing the full commercial version (mailed to you on CD-ROM) can also be found at this Web site.

3.1.2 Installing the Windows Version of OOPIC Pro

To install OOPIC Pro, run the installer program by double-clicking on the icon. Follow the prompts to tell the installer where the application should be placed and where you would like shortcuts to be placed. After the installer has finished, you will be able to run OOPIC Pro using the create shortcuts (typically on the desktop or in the Programs menu).

3.1.3 Installing the Mac OS X Version of OOPIC Pro

The Windows version of OOPIC Pro is available from http://www.txcorp.com/products/oopicpro/. The evaluation version may be downloaded from this Web site; the full commercial version may also be purchased. The latter will be shipped to you on CD-ROM.

To install OOPIC Pro, run the installer program by double-clicking on the icon. Follow the prompts to tell the installer where the application should be placed and where you would like shortcuts to be placed. After the installer has finished, you will be able to run OOPIC Pro as you would any other Mac application.

3.1.4 Installing the Linux Version of OOPIC Pro

The Linux version of OOPIC Pro can be installed easily by running the graphical installer. The installer can be started with the command

```
sh install.bin
```

Follow the prompts to tell the installer where the application should be placed and where you would like shortcuts to be placed. After the installer has finished, you will be able to run OOPIC Pro by running the shell script `oopicpro.sh` located in the `bin` directory. If you use a desktop system such as KDE or GNMOE, you may also want to create icons for desktop or menus. An icon for OOPIC Pro has been provided for your use. Please consult your window manager documentation for instructions how how to create links.

Note that under Linux, OOPIC Pro must be able to open its own xterm windows. Therefore, the TERM environmental variable must be defined. The existence of TERM is generally assured if OOPIC Pro is being run from a graphical environment.

3.2 Overview of Input File Structure and Parameter Groups

This section of the manual provides an overview of the structure and syntax of an OOPIC Pro input file, and gives a brief description of each group of input parameters. A sample input file is shown, and a general procedure for creating input files is outlined. More, detailed information about specific parameters can be found in Sections 4, 5, and 6 of this manual.

An input file is made up of a series of text blocks denoted by a heading and delimited by pairs of curly brackets. The text blocks are processed by the OOPIC Pro input parser to derive a full description of the simulated system.

Input Block Example:

```
Grid              // OOPIC keyword begins the input block
{                 // beginning of parameter definitions
   J = 10
   x1s = 0.00     // each parameter name is a keyword
   x1f = 0.05
   K = 10         // comments like this are allowed in the input file
   x2s = 0.00
   x2f = 0.05
}                 // closing curly bracket marks the end of the block
```

Each line of the file is separated by a carriage return. The use of a semicolon or other character to denote the end of a line will result in a parsing error. The opening and closing brackets must appear alone on separate lines. Comments can be denoted by using "//", similar to C++; note that the parser will interpret the rest of a line as a comment after seeing "//".

3.2.1 Procedure for Creating an Input File

The presence of a single `Description` block is required for the input file. It must appear as the first block in the input file. The `Description` block is normally used as a documentation or comment block At a minimum it must include a Title (see example below). Descriptive comments and the delimiting brackets are optional but strongly recommended.

The heading of the `Description` block can be any text string, but the convention is to make it the same as the input file name. The contents of this block are passed directly through the parser without interpretation, so it is not necessary to denote comments in this block by "//".

```
Title or name of input file.
{
   Comments go here.
}
```

The `Variables` block is completely optional but must follow the `Description` block if it is present. Within the `Variables` block, text strings can be equated to numeric values that can then be used as symbols throughout the remainder of the input file. Although not required, using variables simplifies future modification of an input file makes it more readable.

OOPIC Pro includes the built-in constants PI and e, which can be used in the `Variables` block or at any other point in the input file.

```
Variables
{
   Jmax = 20
   Vb = sin(.56*PI)  // PI is known to the OOPIC Pro parser
   sol = 2.99e8
}
```

A `Region` block will follow the `Description` and `Variables` blocks. Several parameter group blocks must be included inside the `Region` block. The various types of parameter group blocks are described in the next section.

```
Region
{
   // input elements delimited by curly braces
}
```

3.2.2 Parameter Groups Within the Region Block

Input file parameter groups that are specified within the `Region` block can be divided into six different categories. These categories correspond to the different types of physical entities that OOPIC Pro can model.

This section briefly describes these parameter groups. Detailed information about constituent parameters, how they are used and the values they can be assigned is provided below in Section 5 and 6 of this manual. To learn about specific input files for example problems, see Appendix 11 below.

Parameter group categories are:

1. Grid and Control - the parameter groups in this category must appear in the input parameter file.
   (a) Grid - Specifies the physical dimensions of the simulated region, whether Cartesian or cylindrical geometry is to be used, the details of the grid or mesh, and other related parameters.
   (b) Control - Specifies the time step for the simulation, external electric and magnetic fields, whether to use electrostatic or electromagnetic field solve, a seed value for the random number generator and other global parameters.
2. Particle Creation - this parameter group is used to specify how and when particles enter the simulation. If the addition occurs at time zero in the simulation, then this initial presence will have to have distribution parameters supplied for position, velocity, etc in the `Load` segment for each `Species`.
   (a) Species - Specifies the characteristics of a particle type present in the simulation. One `Species` block is included for each type of particle. Different groups of particles are distinguished by giving them different names in different `Species` blocks. This is useful in tracking electrons that derive from different sources.
   (b) MCC - Monte Carlo collision parameters used to model collisions of particles with a background gas as well as the resulting ionization of the gas. Multiple gas types can be supported, each with its own MCC block.
   (c) Load - Parameters specifying the initial spatial and Maxwellian velocity distributions of a particular particle species, which is specified by name.
   (d) VarWeightLoad - Same as `Load`, but uses variable weighting of the macro-particles in order to keep the macro-particle number density uniform across a cylindrical grid. This is achieved by linearly increasing the weighting factor of the macro-particles with the radius.
   (e) PlasmaSource - defines a rectangular region in which a plasma is generated at a constant rate, as well as the initial placement and velocity profile of the plasma constituents.
   (f) Particles can also be created by emission from a surface. This is discussed in the subsection on particle emitters below.
3. Boundary Conditions - OOPIC Pro supports the definition of a number of physical boundary types as listed below. Most of these types require a generic set of parameters to specify geometric or time-dependent properties. For some boundary conditions, complex geometries can be specified by defining multiple Segments. Detailed information for each boundary condition type and use of `Segment` blocks is given in Section 6.3.1, along with examples.

(a) Dielectric - Specifies a dielectric material with arbitrary index of refraction. For electrostatic simulations, charge can accumulate as particles hit the surface. Only one `Segment` specifier allowed.

(b) Conductor - Uses same parameters as Dielectric, but imposes perfect conducting BC's on fields, and multiple `Segment` specifiers are allowed. Any charge or current from particles hitting the boundary is effectively conducted away in zero time.

(c) Equipotential - Behaves like a perfect conductor, which is grounded to a specified potential, which may vary in time.

(d) Polarizer - Applies conducting BC's to the fields, but allows a fraction of incident particles to pass through, based on a specified transmissivity.

(e) Secondary - Specifies how secondary electrons are to be produced at a boundary. Production can be limited to incidence of a designated species.

(f) Secondary2 - A Vaughan-based model, which includes energy and angular dependence and a full emission spectrum, including reflected and scattered primaries, as well as true secondaries.

(g) DielectricRegion - Same parameters as Dielectric, but includes coordinates to specify a rectangular region rather than a simple linear segment.

(h) DielectricTriangle - Same parameters as Dielectric and DielectricRegion but includes coordinates to specify the vertices of a right triangle.

(i) CurrentRegion - Describes the magnitude and spatial distribution of the current flowing within a specified region of the grid.

(j) Iloop - Causes a loop of current to be included in the simulation. It is useful as an external source.

(k) CylindricalAxis - This special boundary condition is necessary for an r-z grid (cylindrical geometry), if the line r=0 is included in the simulation.

4. Ports

   (a) ExitPort - A boundary where electromagnetic waves can exit the grid, with minimal reflection.

   (b) Gap - A wave type boundary condition, where a sinusoidal field variation is applied.

   (c) PortGauss - Launches an electromagnetic wave pulse with a transverse Gaussian profile and a specified longitudinal profile. This Port can only be used in Cartesian geometry.

5. Particle Emitters

   (a) BeamEmitter - Produces a beam of macro-particles of the specified `Species`, with the specified particle current, which can vary in time and space. A great deal of control is provided over beam temperature. Perfect conducting BC's are applied to the fields.

   (b) EmitPort - Same as BeamEmitter, but it does not apply any BC's to the fields.

   (c) VarWeightBeamEmitter - Same as BeamEmitter, but it uses variable weighting of the macro-particles in order to keep the macro-particle number density uniform across a cylindrical grid. This is achieved by linearly increasing the weighting factor of the macro-particles with the radius.

   (d) FowlerNordheimEmitter - Models field emission of electrons from a surface, according to the Fowler-Nordheim theory.

6. User-Defined Diagnostics - There are a number of computed quantities that can be graphically monitored during the course of a simulation.

3.2.3 Example Input File

The following example shows the basic structure of an OOPIC input file. Information regarding the meaning and use of data defined in the parameter groups can be found in Section 5 and 6 below.

```
ExampleInputFile      //  The Description block. The name and contents of
{                     //    this  block  have  no  effect  on  program operation.
   This is an example of an input file.
}

Variables             //  Variables  defined  in  this  block  can  be referred to
{                     //   elsewhere in the input file
    x1max = .1
    x2max = .02
    Jmax = 16
    Kmax = 16
    Cartesian = 1    // specify Cartesian geometry
}

Region               // Region block. All parameter group blocks
{                    //   must be within the Region block
 Grid                      // Grid block - used to specify Region size and
 {                         //   mesh parameters
  J = Jmax
  x1s = 0.0
  x1f = x1max
  n1 = 1.0
  K = Kmax
  x2s = 0.0
  x2f = x2max
  n2 = 1.0
  Geometry = Cartesian
 }

 Control                   // Control block
 {
   dt = 1.0E-11            // specifies time step in seconds
   ElectrostaticFlag = 1   // specifies electrostatic field solve
 }

 Species                   // Species block
 {
   name = electrons        // name of the particle species
   m = 9.11E-31            // physical mass of the particles
   q = -1.6e-19            // physical charge of the particles
 }

 BeamEmitter               // Specifies boundary which emits a beam of particles
 {
   j1 = 0                  // lower endpoint in 1st dimension
   k1 = 0                  // lower endpoint in 2nd dimension
   j2 = 0                  // if j2=j1, then emitter is parallel to 2nd axis
   k2 = Kmax/4             // upper endpoint in 2nd dimension
   normal = 1              // direction of emitter
   speciesName = electrons // must be defined above in Species block
   I = 10                        // Emission current
   np2c = 1.0E7            // numerical weight of the macro-particles
                           //  (\# of physical ptcls per computational ptcl)
   v1drift = 1e7           // speed of particles along 1st dimension in m/s
}

 Conductor                 // perfect conducting boundary
 {
```

```
   j1 = 0                  // This is the boundary condition for
   k1 = Kmax/4             //   the left side of the region
   j2 = 0
   k2 = Kmax
   normal = 1
 }
 Conductor
 {
   j1 = Jmax               // This is the boundary condition for
   k1 = Kmax               // the right side of the region
   j2 = Jmax
   k2 = 0
   normal = -1
 }

 Conductor
 {
   j1 = 0                  // This is the boundary condition for
   k1 = 0                  //   the bottom side of the region
   j2 = Jmax
   k2 = 0
   normal = 1
 }
}  // End of the Region Block
```

4 The Graphical User Interface (GUI)

In this section, we describe the GUI that enables the user to interactively start, stop and restart the application, while dynamically viewing field and particle data.

4.1 Launching the GUI

Unless otherwise specified, the directions and descriptions that follow apply to all supported platforms. Some of the example graphics are taken from the Windows implementation and some from the Linux version. In the discussion below, we assume that the software package resides in a folder or directory called ”oopicpro”. In practice, the name might be somewhat different.

4.2 Providing Input Data

When the executable is invoked an Open file dialog window such as that shown in Figure 1 will appear. This dialog allows interactive browsing to the location of the desired input file. The default folder or directory is /oopicpro/input/, where a number of sample input files are kept. Double click on the input file to be loaded.

When OOPIC Pro is installed, the input directory ”input” is created. OOPIC Pro input files have a *.inp extension. Input files can be created and stored anywhere. If they are not stored in OOPIC Pro’s default input directory, then OOPIC Pro will have to be directed to their location by the standard browse operations provided by the Open file dialog box.

4.3 Using the GUI to Control the Simulation

This section discusses aspects of the GUI related to control of the overall simulation.

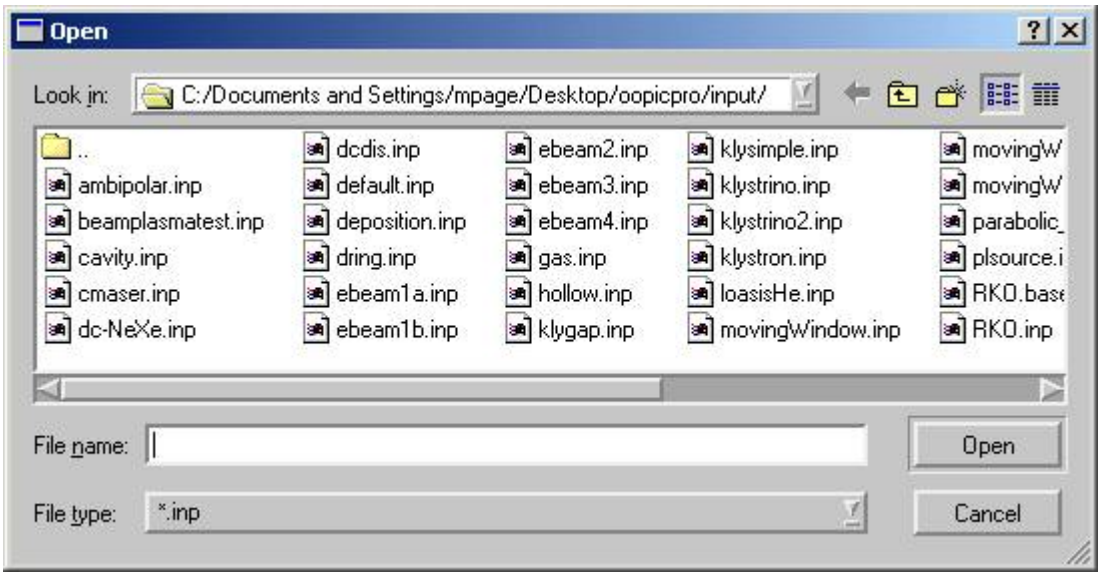

Figure 1: The input file dialog window.

4.3.1 The Control Window

After the OOPIC Pro input file has been specified, the OOPIC Pro control window will appear. The control window consists of a menu bar, a set of four control buttons and two optional, re-sizable log windows. One window is used for writing diagnostic messages (black text) and the other for error messages (red text). Once the simulation input file has been specified, it will be parsed by OOPIC Pro and loaded. Actions taken by OOPIC Pro based upon contents of the input file will be announced in the output log. Any errors will be displayed in the error log.

When the simulation is running the current simulation time is shown on the lower left side of the Control Window border. The name of the current input file can be found on the right side of the border.

4.3.2 The Control Window Menu Bar

The following is a description of the components of the menu bar and their functions. From this point forward, we will use the notation "→ " to denote selection from a sub-menu, for example:

File → Open → Input File...

This menu action is achieved interactively, by clicking File on the menu bar and holding the mouse button down while sliding down the drop-down menu until Open is highlighted. When the hierarchical sub-menu for Open appears, drag to select Input File... and then release the mouse button. If an ellipsis ('...') appears at the end of a menu choice, this means that subsequent action will be necessary - usually the provision of further information through a dialog box or some sort of system query mechanism.

File Menu The File menu presents three menu choices. Two have their own sub-menus:

- Open Input File ... - Open a simulation input file
- Open Dump File ... - Open a previously-saved simulation dump file. This action will fail disastrously, unless the selected dump file was previously generated from a simulation using the same input file that is now active.
- Dump current data ... - Save the current state of the simulation in a designated dump file. The default name will be the base name of the input file, with ".dmp" extension.

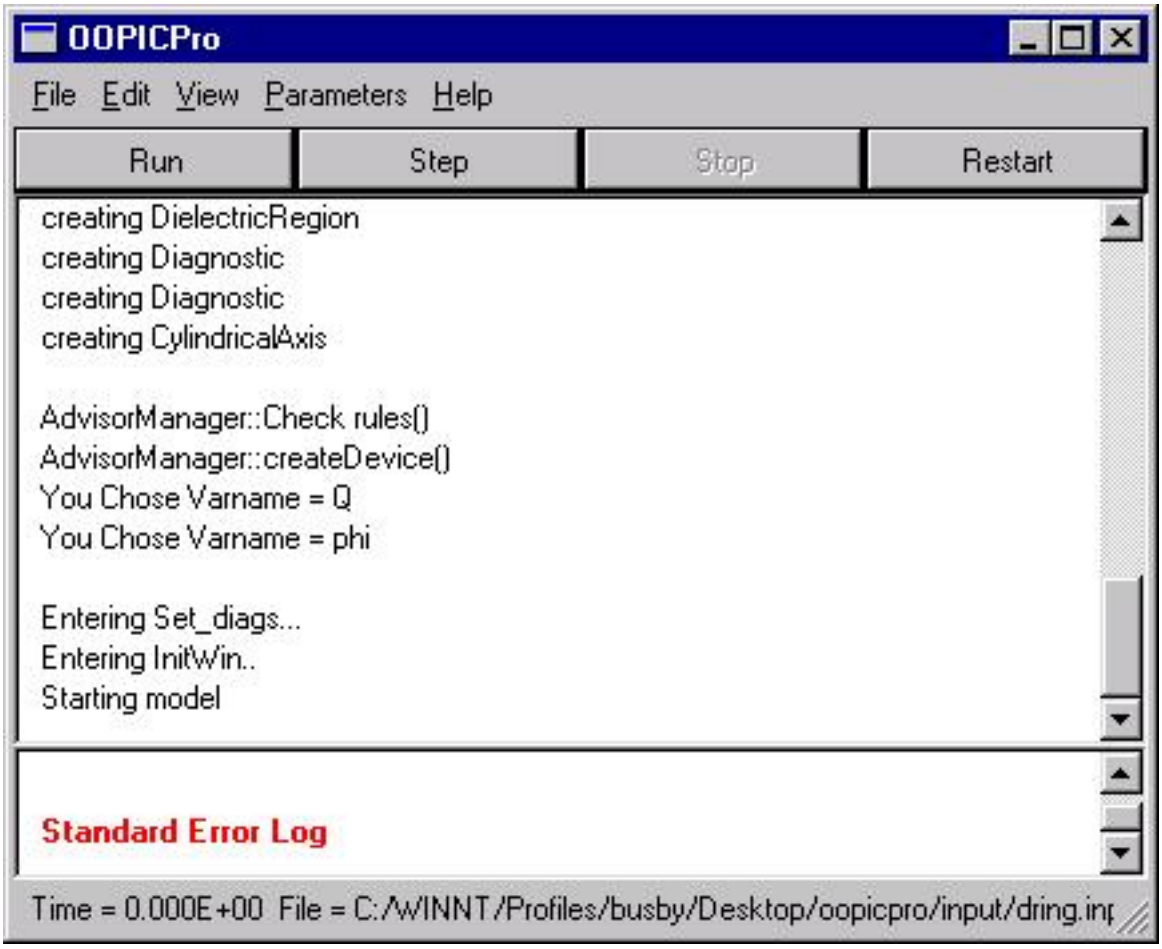

Figure 2: OOPIC Pro Control Window

- Save movie config file ... - Save movie configuration information.
- Save image sequence ... - Allows user to save a series of images from the diagnostics which can then be combined into an animation.
- Exit - Quit OOPIC Pro

An input file can be loaded only if the current simulation is stopped. Loading another input file eliminates the current simulation and starts a new one.

The OOPIC Pro session can also be terminated immediately by closing the control window.

Edit Menu From the Edit menu, selecting Input text file.... displays the current input file. Here it can be viewed or modified You can also save the current file or save a copy of the file.

Parameters Menu The Parameters menu presents three menu choices. Each has it's own sub-menu:

- Edit input file ... — Opens a text edit dialog box where the current input file can be viewed or modified.
- Run configuration ... — Opens a dialog box where the user can edit the run parameters of the simulation.

View Menu

- Diagnostics ... — Opens dialog box where the user can select diagnostics to display
- Log
  - Standard Output — Toggles display of standard output log
  - Standard Error — Toggles display of standard error log

  Both the Output Log and the Error Log are present at startup by default.

  The output log is located in the upper part of the interior of the main control window. It displays comments on the status of the program, and gives messages as each component of an input file is interpreted and added to the simulation.

  The error log is located in the lower part of the interior of the main control window. It alerts the user to errors in the simulation that may mean that the simulation should be stopped.

Help Menu

- OOPIC Pro manual — Opens the User's Manual using the system's default PDF viewer
- About OOPIC Pro — Displays copyright and version information

4.3.3 The Control Window Control Buttons

The four large buttons below the menu bar control the execution of the simulation.

- The [Run] button begins or continues a simulation.
- The [Step] button advances the simulation one frame. Note that this will not be the same as advancing the simulation one time step if the Number per Update parameter has been changed in the Parameters→Run Configuratin... dialog. Changing the Number per Update parameter has the effect of letting the simulation run for multiple time steps between screen/graphics updates. Therefore the [Step] button actually advances the simulation to the time of the next screen update.

Figure 3: Example Diagnostics Display Selection List

Stop stops the simulation.

Restart resumes the simulation from initial conditions.

4.4 Displaying Diagnostic Plots

Upon opening an input file in OOPIC Pro, there will be one or more position plots visible, one for each of the species present in the simulation. These plots are examples of default OOPIC Pro diagnostic plots.

OOPIC Pro can also display plots of other diagnostic quantities. Some standard diagnostics are automatically included. Additional diagnostics can be specified in the input file by the user (See Section 6.6).

To display the plot of a diagnostic, select the checkbox next to thediagnostic name. A window containing a list of the available diagnostics will appear. Additionally, you can open or close all available plots using the “Open all” or “Close all” buttons on the right of the dialog box.

When you are finished selecting plots, choosing “Done” will close the dialog box.

4.4.1 Default Diagnostics

A specified set of diagnostics is available for display by default. Additional diagnostics may be requested according to specifications in the input file. The format for requesting additional diagnostics is given in Section 5.6.

The following diagnostics are available by default (in the order they appear in the diagnostics window). Refer to the definition of parameters in the `Control` specifier for further information on the parameters that control the inclusion of components in this list. The labels used for the designation of coordinate and diagnostic components will be explained in detail later. For example, let the designations 1, 2, and 3 relate to the coordinate axes of the coordinate system used to define the simulation.

- As a function of x1, x2 and x3:
  - E1, E2 and E3
  - B1, B2 and B3
  - I1, I2, and I3 if ElectroStaticFlag is not set
  - Ue and Ub
- As a function of x1 and x2:
  - Poynting Vector if ElectroStaticFlag is not set
- As a function of x1, x2 and x3:
  - Rho
  - Densities of any neutral gases present
  - Boltzmann Rho if BoltzMannFlag is set
  - Total Rho if BoltzMannFlag is set
  - Phi if ElectroStaticFlag is set
  - Divergence Error if ElectroStaticFlag is not set
- Position plots for all species
- Phase space plots for all species for all combinations of position vs. velocity:
  - U1 vs. x1
  - U2 vs. x1
  - U3 vs. x1
  - U1 vs. x2
  - U2 vs. x2
  - U3 vs. x2
- As a function of x1 and x2:
  - E
  - B
  - I, if ElectroStaticFlag is not set
- Time domain plots
  - If IdiagFlag is set
  - I history for boundary
  - I local history for boundary

- If EFFlag is set
  - Poynting history for boundary
  - Poynting local history for boundary
- Composite plot of Poynting history
  - Potential and kinetic energies
  - E and B field energies
- Time domain plots
  - Average magnitude of divergence error if ElectroStaticFlag is not set
  - Number
  - Kinetic Energy
  - Total density
- RMS beam parameters for species with rmsBeamSizeFlag set
  - Average beam size
  - RMS beam size
  - Average velocity
  - RMS velocity
  - RMS emittance
  - Average energy
  - RMS energy
- Average kinetic energy vs. time
- Number density for each species
- Initial number density for each species if Show_Loaded_DensityFlag set
- Boundary diagnostics as defined by user

4.5 The Various Plot Types

To display diagnostics, OOPIC Pro uses several types of plots. Each is appropriate to the diagnostic quantities being plotted.

4.5.1 2-D Particle Plots (configuration space)

A 2-D particle plot (see Figure 4) shows macroparticle positions within the simulation region. A plot of this type is automatically displayed for each species included in the simulation after the program starts. The axes are either x and y or z and r, depending on the geometry specified for the region in the input file (Geometry keyword in the `Grid` block). These plots also show the exterior and interior boundaries defined for the simulation. Each type of boundary (conductor, dielectric, emitter, etc) is represented by a unique color.

In this plot, the boundary defined to be the `BeamEmitter` in the input file is drawn in brown. Conductors are drawn in green. The rightmost Conductor boundary does not draw since it corresponds to the limit of the graphics window. This inaccuracy can be fixed by setting a new, larger value for the graphics window maximum via View→ Options and entering a new value in the axis maximum value text box.

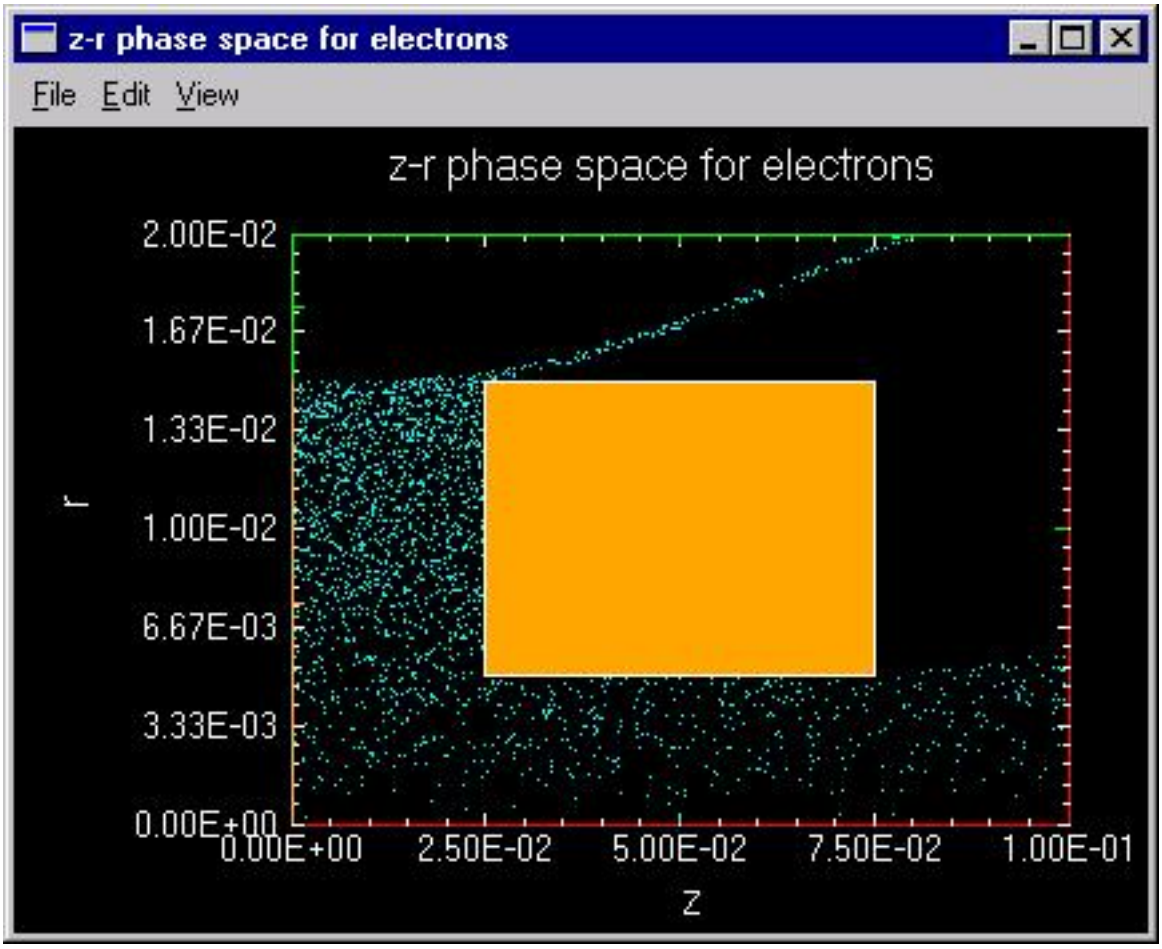

Figure 4: A 2-D Particle Plot

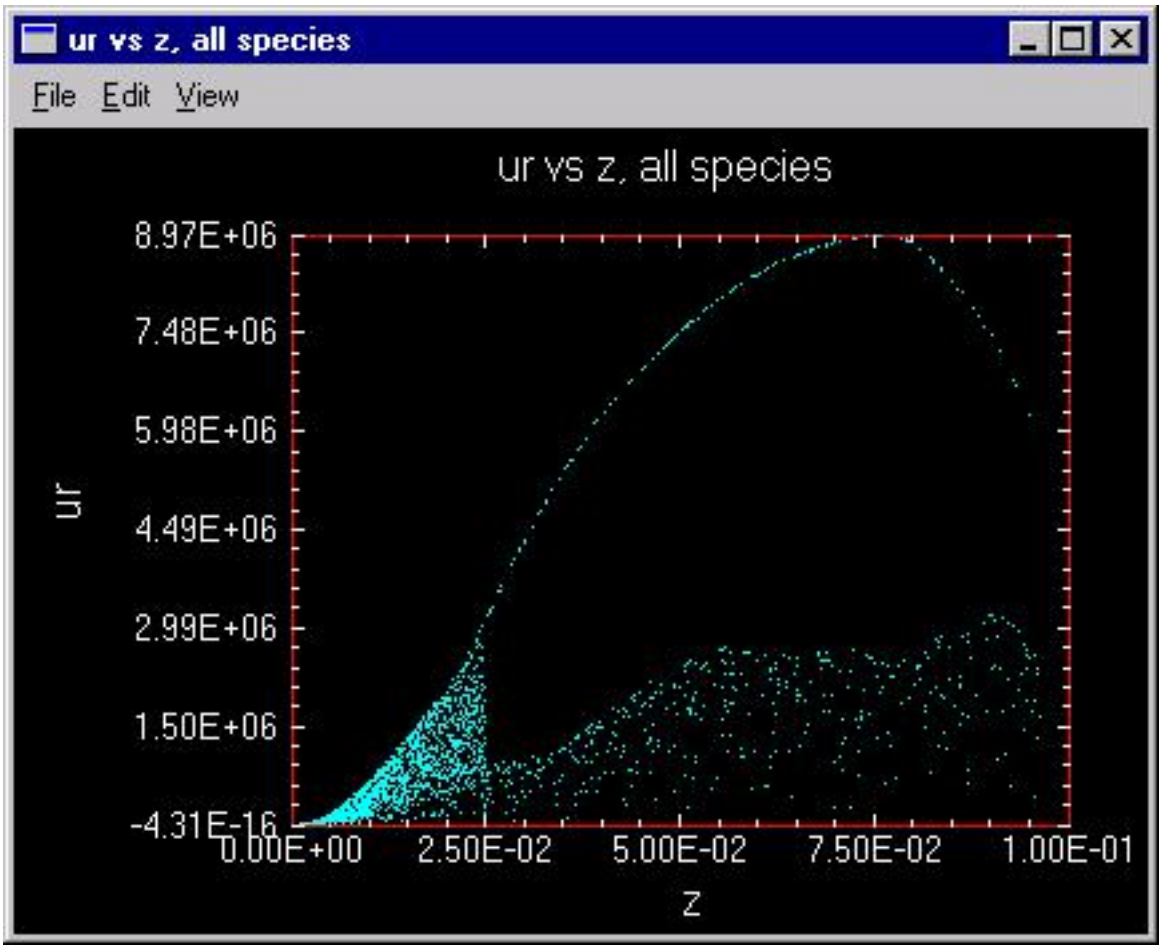

Figure 5: A 2-D Phase Space Plot

4.5.2 2-D Phase Space Plots

2-D phase space plots show particle location versus velocity. Specifically, data for the following plots are automatically collected and the plots are listed in the Diagnostics menu as:

x1 vs. u1, x2 vs. u1

x1 vs. u2 x2 vs. u2

x1 vs. u3 x2 vs. u3

where u[i] is gv[i], with v[i] being the velocity component of a macroparticle along the *i*th dimension.

The convention throughout the OOPIC Pro program and documentation is:

x1 is the x or z direction, depending on the coordinate system used.

x2 is either y or r.

x3 is either z or f, the dimension ignored in the simulation.

All species present in a simulation are represented on the same phase space plots.

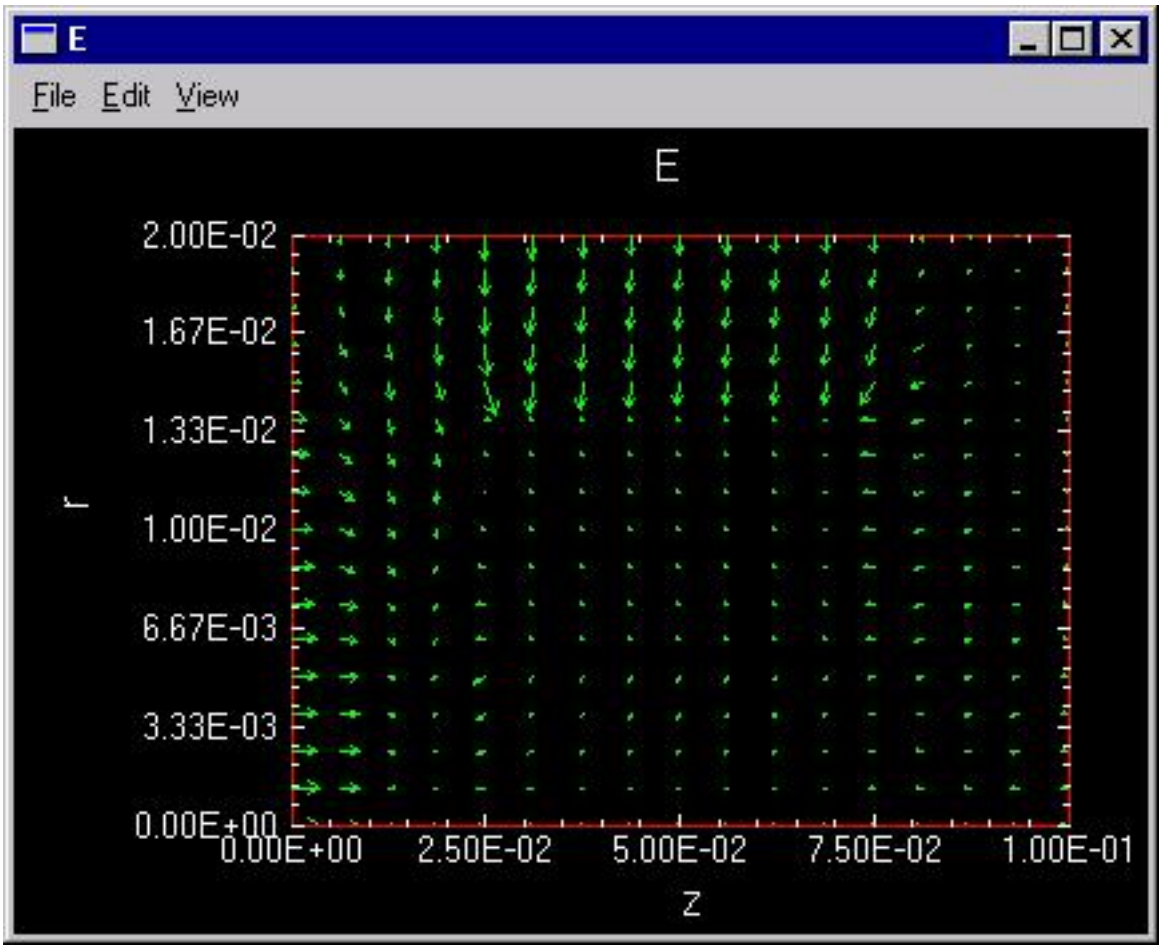

Figure 6: A 2-D Vector Plot

4.5.3 2-D Vector Plots

2-D vector plots can be used to show EM field directions and magnitudes.

4.5.4 3-D Surface Plots

A 3-D surface plot of the scalar value over the region can be drawn for certain parameters. Examples of diagnostics plotted in this way are the components of the electric and magnetic fields, E[x], B[f], etc.

4.5.5 2-D Line Plots

A 2-D line plot shows a time history of scalar quantities. The default diagnostics which are plotted this way are instantaneous total and kinetic energy of the simulation, average kinetic energy, and the number density of various species. Some plots display than one quantity.

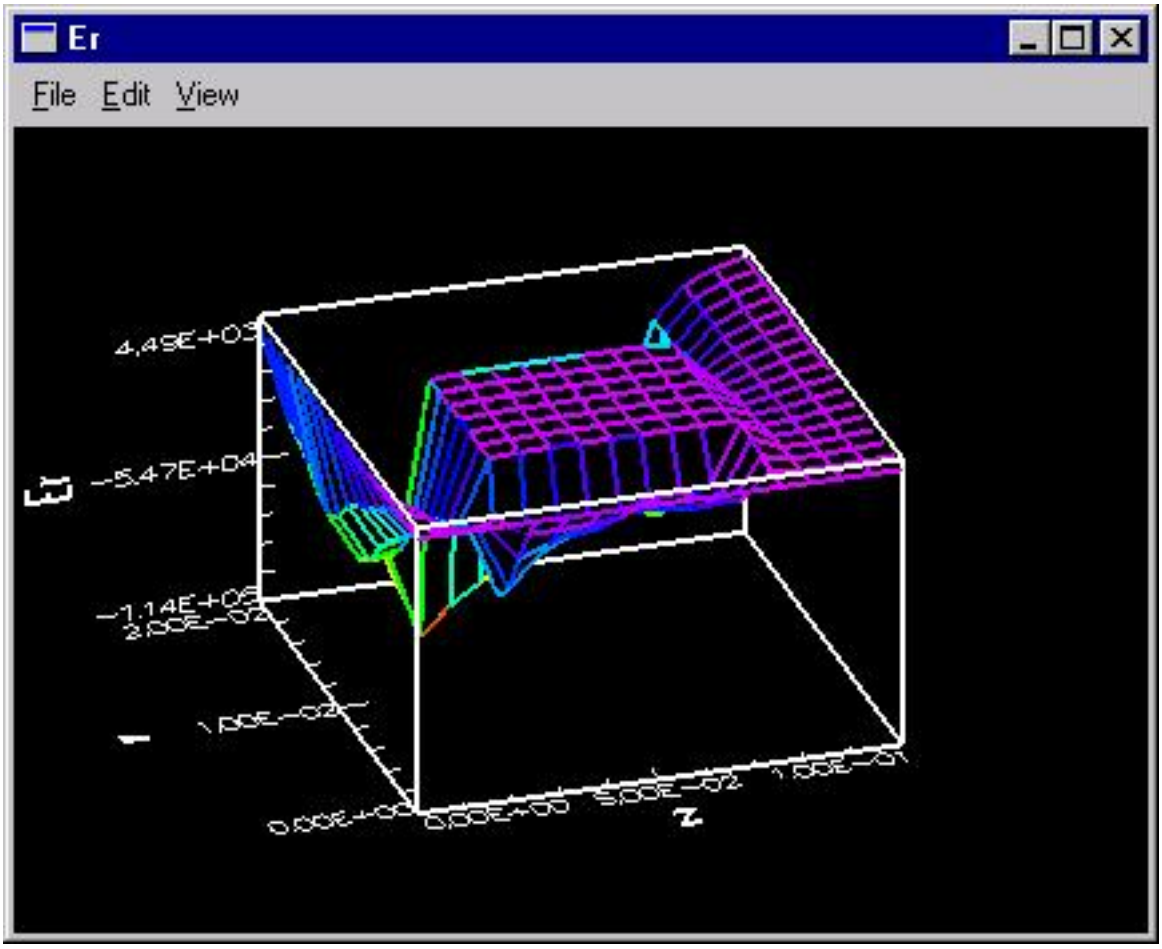

Figure 7: A 3-D surface plot

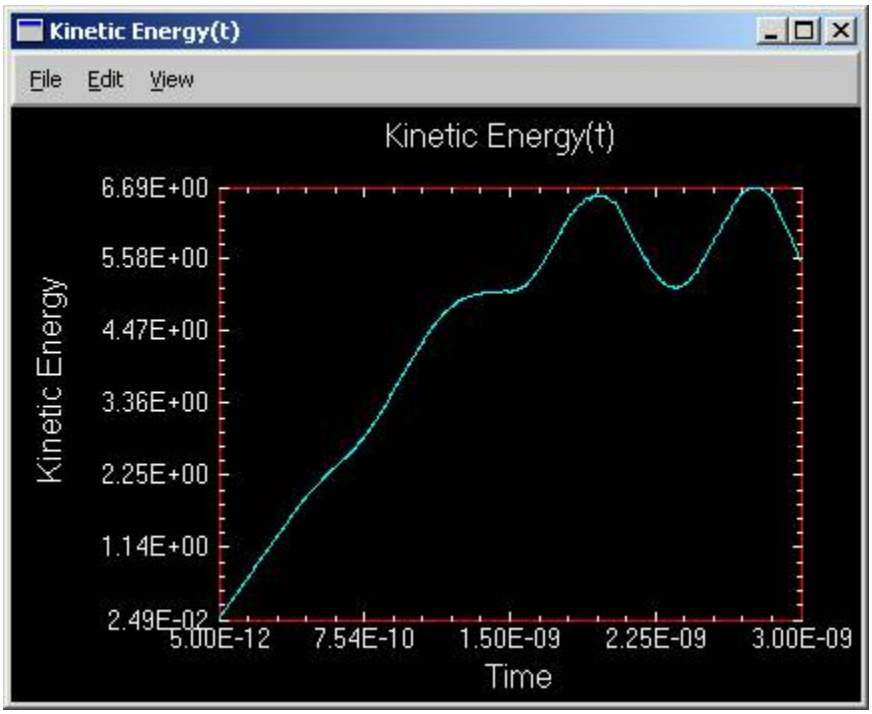

Figure 8: A 2-D Line Plot

4.6 Manipulating Diagnostic Plots

4.6.1 2-D Plots

Data presented by OOPIC Pro's two-dimensional diagnostics plots can be accessed and manipulated as outlined below. Standard methods of transferring the graphics to other applications are supported. In addition, data can be saved in non-graphic file formats that be accessed by other applications and the visual images can be dynamically manipulated.

File Menu Contents of the 2D plot windows can be saved in graphic or text format, printed, previewed or closed from the File menu.

- Save as... — Saves the contents of the diagnostic window in a variety of graphics formats.
- Dump data... — Saves a description of the data displayed in a text format which can be analyzed or post-processed. A text dump will create a considerably smaller file than an image file. (Note that the plot dump files are in text format, while simulation dump files are saved in binary format.)
- Print preview... — Print Preview will show what the active plot will look like on a printer chosen from the standard Select Printer dialog box.
- Print... — Brings up a dialog box that allows formatting and printing of the current plot.
- Close — Close will dismiss the plot window. It can be viewed again by selecting it from View→Diagnostics.

Edit Menu The Edit menu of the plot window contains a single entry that used is to copy the contents of the plot window to the clipboard (Windows only). From the clipboard, the window contents can be pasted into other applications that support graphic content.

View Menu There are four basic manipulations that can be applied to the 2D graphics windows that will alter their appearance.

Crosshair Choosing Crosshair from the View menu will cause the cursor to be replaced by a crosshair. By positioning the crosshair within the plot window and clicking the left mouse button, x and y values for selected data points can be determined. The x, y data pair for the selected point is displayed at the bottom of the plot window.

Trace Choosing Trace from the View menu will cause points plotted in the plot window to not be refreshed for each graphics update interval (Parameters→ Iterations→ Number per Update ...). For example, normally if a plot of particle position is displayed against spatial coordinate axes, the entire assembly of particle positions is erased and replotted for each graphics update interval. With the trace feature turned on, the erasure of particle positions does not occur and the trajectory of each particle can be seen as a continuous curve on the plot.

Zoom In/Zoom Out It is possible to zoom in to smaller areas of a 2-D plot. Choose Zoom in... from the View menu at the top of the plot window.

Position the cursor (an arrow for Windows, a pointing finger for Linux) at the desired upper left corner of the plot. Click and hold the left mouse button, and drag the cursor down and to the right to the desired lower right corner of the plot and then release the mouse button. The plot will then be redrawn with these new limits.

Selecting Zoom out from the View menu and clicking the mouse with the cursor located anywhere on the plot will result in the plot returning to the previous plot limits. If the plot has been zoomed in more than once, then multiple clicks will be required to return to the original plot limits.

The simulation must be resumed or restarted after the zoom operations are completed. This will also restore the default plot limits.

Options The basic appearance of 2D plots can be controlled through the functions provided on the Options menu. Selecting Options from the View menu on a 2D plot produces a selection box with three tab choices available. The tabs are labeled Axes, Title and Tick Marks.

Selecting the Axes tab allows the plot coordinate minimums and maximums to be explicitly set for both axes. Optionally, the axes can be autoscaled, based on range and the coordinate labels can be written in scientific notation (default) or floating point format. The scale types can also be set to either linear or logarithmic.

Selecting the Titles tab allows the labels attached to the coordinate axes to be changed as well as the color of the label text.

Selecting the Tick Marks tab offers the option of drawing tick marks inside or outside the plot area and allows the color of the tick marks to be set.

The options editor window can also be made to appear by pressing the right mouse button while over the plot.

4.6.2 3-D plots

Manipulation and management of data presented in OOPIC Pro's three dimensional diagnostics plots is very similar to the same processes used for two-dimensional data.

File Menu The File menu for 3-D plots window is the same as for the 2-D plots. See Figure **??**.

Edit Menu The Edit menu for 3-D plots window is the same as for the 2-D plots. See Figure **??**.

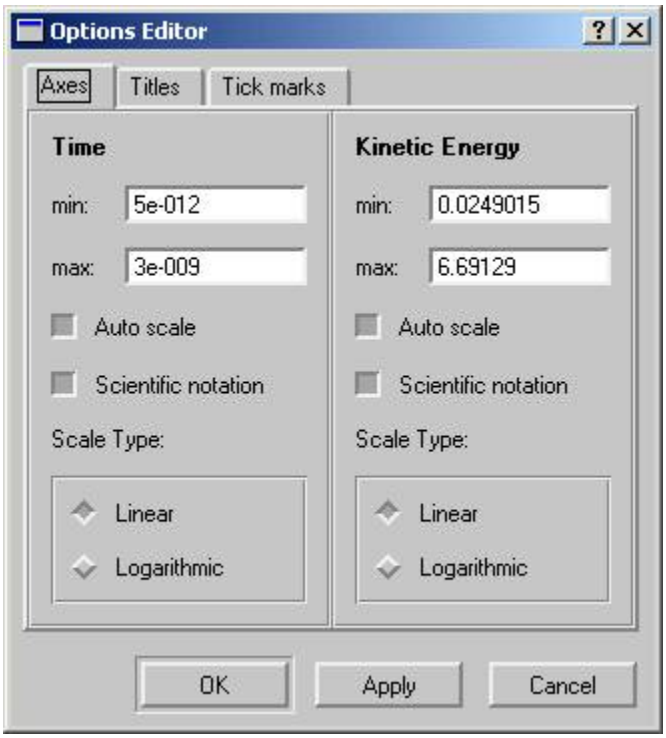

Figure 9: Options Editor window for a 2D diagnostic plot

View Menu The View menu for 3D plots differs significantly from what is offered for 2D plots. The first four choices in the 3D menu are for turning various graphical characteristics of the plot on or off.

- Wire Frame — Wire Frame is on by default and simply shows all six edges of the box that contains the data being plotted as white lines.
- Shading — Shading, which is off by default, gives a color to each panel between grid lines depending on the value of the plotted quantity. Reds and yellows denote low values; purples and blues denote high values.
- Grid — This option draws lines between grid points. If shading is on, the grid lines are black. If not, they take on colors in the same fashion as shading.
- Invert B/W — Invert B/W changes the background of the plot to white. Note that the background is white when printed, regardless of the state of Invert B/W.

CrossHair The CrossHair option is disabled for 3-D plots.

Modify Viewpoint Selecting Modify Viewpoint causes the Viewpoint Controls window to appear. The viewpoint can be changed, i.e. the figure can be rotated around three axes. Zooming is also allowed within the limits of one-tenth to four times the original size. The Viewpoint Controls window will initially appear in front of the plot window but can be moved by clicking and dragging the window on its top bar.

To alter the rotated view of one of the axes, click on a tab corresponding to an axis, and type a new value for the rotation, or click and drag the pointer next to the hashed line until the plot appearance is satisfactory. The zoom factor for the plot can also be changed by clicking on the Zoom tab.

Choosing Options will cause an Options Editor menu to appear with four header tabs. The first three of the tabs provide the same functionality for 3D plots as they do for 2D plots.

Selecting the fourth tab, Grid, allows the adjustment of characteristics of the plot grid. The grid is turned on or off by a selection available directly on the View menu.

As with 2D plots, the options editor window can also be made to appear by pressing the right mouse button while over the plot.

4.7 Dump Files

The current state of a simulation can be saved in a dump file. There are alternatives for the format in which the dump file is saved. The default file format will be HDF5. This is the case for Linux and Mac OS X systems. The Windows version of OOPIC Pro does not currently support the HDF5 file format for dumps. Instead a special OOPIC Pro binary file format is used. This special format is also available as a dump file format on Linux and Macintosh systems.

4.7.1 Saving a Dump File

To save a dump file, select File→ Save dump file...

4.7.2 Saving a Dump File Periodically

OOPIC Pro can also perform automatic saves a of simulation state at regular intervals. To periodically save system status (checkpointing), open the Run Configuration dialog by choosing Parameters→Run Configuration from the main OOPIC Pro window. The number of iterations per checkpoint data dump can be changed in this dialog, as well as the number of iterations per plot update — note that these can be different values. Each time the dump file saves it will overwrite any previous dump files. The name of the dump file can be specified at the beginning of the simulation by saving a dump file normally as described above. Otherwise, the name of the dump file will assume the default value. To save in binary format, the dump file extension must be specified to be '.dmp', otherwise the save file format will be HDF5 and the filename extension will have to be specified as '.h5'.

4.7.3 Loading a dump file

To restore a simulation to the state at which it was when the dump file was saved, choose Open dump file... from the main OOPIC Pro window.

4.8 Movies

OOPIC Pro supports the creation of animated sequences of data by replaying sequential snapshots of the same data display.

4.8.1 Creating Movies

OOPIC Pro has the capability of saving multiple frames of one or more plots during a simulation for sequential viewing later. To create an OOPIC Pro movie, open the Image Sequence Parameters dialog by choosing File→Save image sequence... from the main OOPIC Pro window.

By default, OOPIC Pro will place the image files in a directory with the same name as the input file. If a movie is being created for a simulation for the first time, a message box will appear informing the user that a new directory '`<input file>_movie`' is being created, where `<input file>` is the name of the input file without the '.inp' extension. Select OK, and a dialog window will appear. The default image file stub appears in the File Name text input box. The default stub is the input file name followed by an underscore. Change the file stub if desired. Then

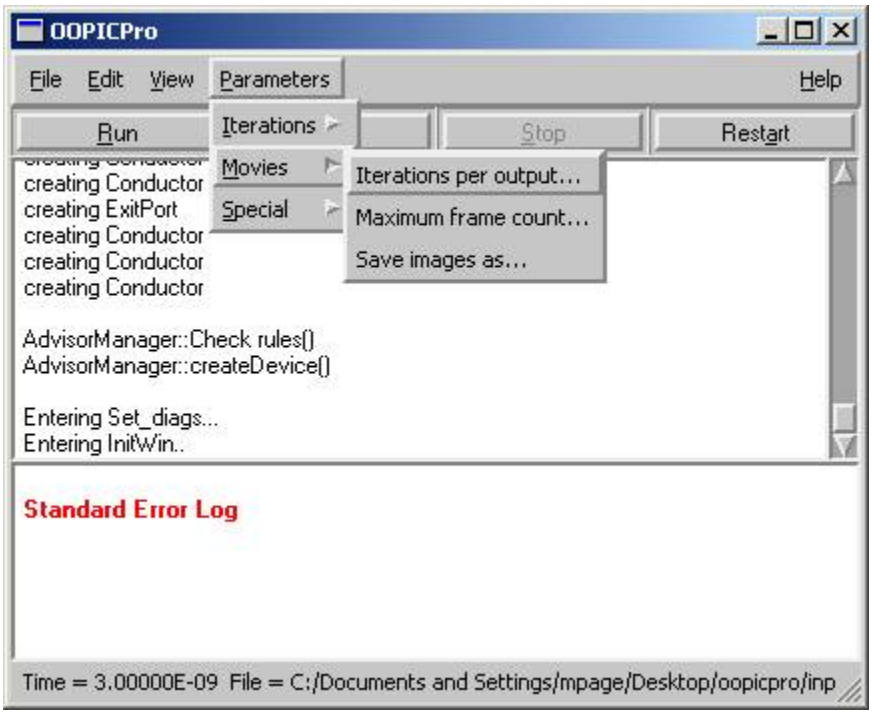

Figure 10: Saving Simulation Output as a Movie

choose the format of the movie image files. The actual image file names will consist of the image file stub, followed by the name of the plot being saved, followed by the frame number, with an image file extension. For example:

```
cavity_z-r_phase_space_for_electrons.00001.png
```

would be the first image file in a movie being created for a simulation titled 'cavity_z-r_phase_space_for_electrons'. The image files will all be saved in the PNG format.

Note: If more than one movie of the same simulation is to be created, each image should have a unique filename. OOPIC Pro will NOT automatically delete previous image files with the same stub before making a movie. Failing to remove old files with the same stub will result in having extra frames from previous movie.

After choosing an image file stub, the frame rate of the movie should be specified. Select Parameters→ Movies→ Iterations per Output... The number displayed in the value entry window specifies the number of simulation plot frames (not time steps) per movie frame.

Similarly, the maximum number of frames in the movie can be specified by changing the Maximum frame count parameter. Its default is '0', denoting no limit to the number of frames saved.

If the simulation has multiple graphic plots open, a set of image files will be created from each plot window. Rather than load all of these files into the movie display program sequentially, the user can specify a configuration file, which will track the movies from various plots.

4.9 Common problems

Program exits immediately upon attempting to open an input file

It is likely that there has been a parsing error while reading the current input file.

Check the input file for bad syntax. If the error can't be found, remember that the only elements necessary for OOPIC Pro to start are the `Grid` *and* `Control` *elements. Delete all elements but these from the input file. Restore*

Table 1: Allowed Data Types in OOPIC Pro Input Files

Data Type	Usage
int	integer
string	Any set of characters with no whitespace
scalar	floating point single or double precision number in either decimal or scientific notation
flag	Binary flag, 0 or 1

them to the input file one at a time until the error recurs.

3D plots aren't centered in the diagnostic window

This is a problem with the Mesa library. Forcing the diagnostic window to refresh by changing its size works around this error.

5 Input Files, Part I - Large Scale Issues

In this section the basic structure and syntax of an input file is explained.

The following nomenclature is used to discriminate among the various levels of the input file: The input file is made up of blocks (`Description`*,* `Variables`*,* `Region`*), blocks are made up of elements (*`Dielectric`*,* `Conductor`*, etc.) and elements may contain specifiers (*`Segment`*). The numeric or symbolic value of parameters can assigned at either the element or specifier level depending on context and usage.*

5.1 Structure

An OOPIC Pro input file consists of three major blocks of information, one of which is optional. as noted in Section 3.2.1, The `Description` *block is required but the* `Variables` *block is optional. The* `Region` *block is required by OOPIC Pro to fully specify the physical parameters of the simulation. A discussion of the content and usage of each type of block will follow a discussion of the input file parser syntax rules.*

5.2 Syntax

The syntax rules implemented by the OOPIC Pro input file parser provide for a variety of data types, a number of arithmetic operators, symbolic constants and special functions. High-level details of the parser are given in the next sections.

5.3 Data types

There are four data types allowed in an OOPIC Pro input file:

5.4 Floating Point Notation

Both decimal and scientific notation are allowed.

All of the following are understood by the parser and are equivalent.

0.03

.03

3e-2

3E-2

3e-02

5.5 Operators

Table 2: Operators in OOPIC Pro

Operator	Description
=	assignment
-	subtraction
+	addition
	multiplication
/	division
%	modulus
NEG	negation (unary minus)
^	exponentiation

5.6 Constants

Table 3: Constants in OOPIC Pro

Operator	Value	Notes
PI	3.141592654	must be capitalized; parser is case-dependent
e	2.18281828	

5.7 Functions

The OOPIC Pro parser implements the following set of special functions:

Table 4: Special Functions in OOPIC Pro

Function	Description
sin()	sine of argument
cos()	cosine of argument
atan()	arctangent of argument
ln()	natural logarithm of argument
exp()	natural exponential of argument
sqrt()	square root of argument
pow(x,y)	x to the yth power
pulse(t,tdelay,trise,tpulse,tfall)	finite pulse (see Time Dependent Functions)
step(x)	returns 0 if x $<$ 0 and returns 1 otherwise
ramp(x)	returns 0 if x $<$ 0 and returns x otherwise

5.8 Time Dependent Functions

Boundaries can defined that exhibit time-dependent behavior. For example, `Equipotential` *boundaries can have a time-dependent voltage, and* `BeamEmitters` *can have time-dependent currents.*

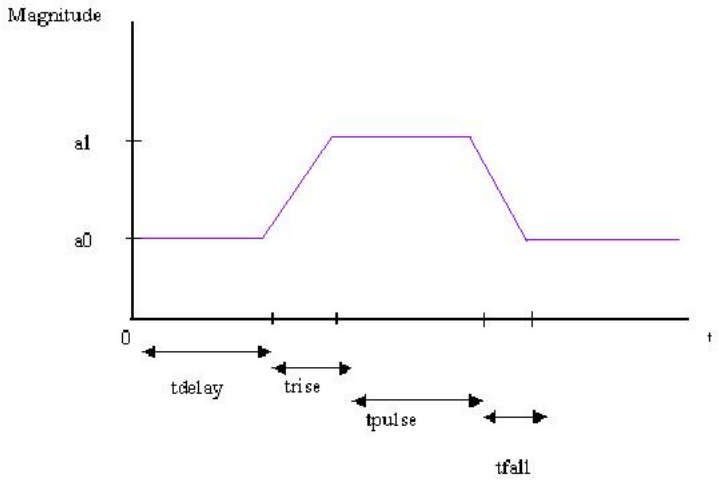

Figure 11: Diagram of Time Dependent Envelope Function

xtflag is a boundary parameter that applies to all types of boundaries implemented in OOPIC Pro. This is a generic boundary parameter meaning that it is used in the specification of all OOPIC Pro boundary types including conductors, equipotentials, emitters and any other parameter group belonging to the boundary class. From the point of view of an object-oriented-software implementation this means that the Boundary class is a parent to many other types of classes implemented in OOPIC Pro and that xtflag is parameter that is inherited by the classes derived from this parent class.

The default value for xtflag is 0. This implements time-dependence of the form:

$$F(t) = Envelope(t) \cdot Sinusoid(t)$$

where t = simulation time.

The envelope function is a piecewise linear function whose parameters specify a function like that shown in Figure 11.

The envelope is completely parameterized by specifying the values for a0, a1, tdelay, trise, tpulse and tfall.

The sinusoid is parameterized by values for: phase, frequency, A and C in the function:

$$F(t) = A \cdot sin(2 \cdot \pi \cdot frequency \cdot t + phase) + C$$

5.9 String Time Functions

*If the characterization of a boundary condition doesn't fit in the Sinusoid(t)*Envelope(t) template provided above, a more general function using any of the operators and functions understood by the OOPIC Pro parser may be used instead by altering the value of xtflag. This function is a string and should be assigned to the parameter F. F is also a generic boundary parameter.*

If xtflag is set to 1, then the string value for F will be interpreted as a function of t (time).

For other values of xtflag:

xtflag = 1 F = f(t) Time-dependent sine function

xtflag = 2 F = f(x) Position-dependent function

*xtflag = 3 F = f1(x)*f2(t) Product of space- and time-dependent functions*

Table 5: xtflag Descriptions

xtflag	Meaning of F
0	Envelope * Sinusoid
1	F = f(t)
2	F = f(x)
3	F = f1(x) * f2(t)
4	F = f(x,t)

xtflag = 4 F = f(x,t) Space-time-dependent functions

Examples of time dependent functions (xtflag = 1):

$F = sin(t)^2$

$F = 10e9 * (t\%1e - 9)$ *(% is the mod operator)*

$F = exp(t\%1e - 10)$

In addition, these special functions are available for the definition of pulse functions:

pulse(t,tdelay,trise,tpulse,tfall) similar to the envelope function above

step(x) returns 0 if $x < 0$ *and returns 1 otherwise*

ramp(x) returns 0 if $x < 0$ *and returns x otherwise.*

5.10 Input File Blocks

At the highest level, the structure of an input file consists of a required `Description` *block, an optional* `Variables` *block and a required* `Region` *block. Section 4.2 explores the basics of the use and definition of information contained in these blocks. More detail is provided for the parameter groups within the* `Region` *block in Section 5.*

5.10.1 `Description` Block

The `Description` *block is a required part of the input file even though it has no impact on the simulation. The name of the* `Description` *block is usually chosen to be the same as the name of the input file itself but this is not a requirement. The purpose of the* `Description` *block is to provide explanatory information to anyone who might read the text form of the input file. Its best use is to provide a robust description of the collective simulation described in the* `Region` *block and a broad overview of the geometry of the simulation, the species involved, interesting properties that are being investigated in the simulation, etc.*

Input Block Example:

```
<Description block name>
{
   .
   .
comments
   .
   .
}
```

5.10.2 `Variables` Block

A `Variables` *block, positioned immediately before the* `Region` *block, can contain user-defined symbolic variables. Its use is optional but recommended because this block is used to create definitions referenced later in the body of the*

input file. Unique string variables are defined here and are assigned numeric values by the '=' operator. Subsequent reference can be made to the symbolic variable anywhere in the remainder of the input file and a numeric equivalent will be assigned in its place by the input parser. The resulting expression is evaluated by the parser. The symbolic variables are more quickly understood and associated with their physical significance in the simulation than just a string of decimal digits.

Input Block Example:

```
Variables
{
  Jmax = 200          // number of cells in x1 direction
  Kmax = 120          // number of cells in x2 direction
  k_half = KMAX/2     // middle cell in x2
}
```

OOPIC Pro will interpret any occurrence of these strings within the `Variables` block as having the values given, if the occurrence is on the right-hand-side of an assignment ('=') operator.

5.10.3 `Region` Block

Parameter group definitions occur within the `Region` block. Numeric values or their symbolic equivalents for all parameters will be found in this block. This should not to be confused with the symbolic variable definitions that appear in the `Variables` block. The definitions made within the `Variables` block are utilized within the `Region` block as a way to make the parameters in use more understandable.

The curly bracket denoting the end of the `Region` block is always located on the last line the input file.

6 Input Files, Part II - Parameter Groups

The following sections define all of the parameters currently supported by OOPIC Pro. The format of entries in these sections is as follows:

Table 6: Format example

parameter	*default value*	*type*	*units*	*description*

where type is either flag, int, scalar or string, and units describes the physical units used, if needed. Units are always MKS unless specified otherwise. For the format of the `Region` block within an input file refer to the example input file provided in Section 3.2.3 or any of the example input files included in the OOPIC Pro distribution.

6.1 `Grid` and `Control`

Here, we describe the `Grid` and `Control` parameter groups.

6.1.1 `Grid`

The `Grid` element is used to specify the size, shape, geometry and other characteristics of the numerical grid superimposed upon the simulated system. It establishes a correspondence between the numerical grid and the geometrical parameters of the system.

The grid parameters listed here without default values are required to be specified in the input file. A `Grid` element will always appear in the input file.

Note: In this manual the names of the coordinate axes are referred to as x1, x2 and x3. If the Geometry flag in this input element is set to a value of 1 then the simulation is computed for a Cartesian coordinate system where x1 corresponds to the x direction, x2 to y and x3 to z. If Geometry is set to 0 then the simulation is computed for a right cylindrical coordinate system where x1 corresponds to z, x2 corresponds to r and x3 corresponds to phi. In both systems, x3 is the coordinate of symmetry and can be ignored.

Table 7: `Grid` Parameters

Parameters	Default value	Data type	Units	Description
`J`	None	int		Number of cells in the x1 direction.
`K`	None	int		Number of cells in the x2 direction.
`x1s`	None	scalar	m	Lower coordinate in the x1 direction.
`x1f`	None	scalar	m	Upper coordinate in the x1 direction.
`x2s`	None	scalar	m	Lower coordinate in the x2 direction.
`x2f`	None	scalar	m	Upper coordinate in the x2 direction.
`Geometry`	0	flag		Cylindrical (0) or Cartesian (1) geometry
`PeriodicFlagX1`	0	flag		Non-periodic (0) or periodic (1) boundary conditions in the first direction. (Applicable to 1-d problems)
`PeriodicFlagX2`	0	flag		Non-periodic (0) or periodic (1) boundary conditions in the second direction
`dx1`	1	string		Function that will modulate the mesh spacing in x1 direction. It is a function of j; so dx1="f(j)" forms are acceptable. Examples: dx1 = j (1/2 < j < J-1/2) or dx1 = j/J + 0.5 (1/2 < j < J-1/2)
`dx2`	1	string		As above; except for k.

Input Block Example:

```
Grid
{
   J = 30
   x1s = 0.0
   x1f = 0.05
   K = 30
   x2s = 0.0
   x2f = 0.01
}
```

6.1.2 `Control`

The `Control` element is used to specify the time step for the simulation, initial static E and B field parameters, a default seed value for random number generator and other simulation control parameters as noted.

Table 8: Control Element Parameters

Parameters	*Default value*	*Data type*	*Units*	*Description*
`dt`	*1E-12*	*scalar*	*sec*	*Simulation time step*
`emdamping`	*0*	*scalar*		*Damping value for high frequency noise. 0 < emdamping < 1*
`movingWindow`	*0*	*int*		*Set to 1 to enable "moving window" feature. This will move fields and particles in the simulation one cell to the left at the speed of light with each time step. Boundaries are NOT moved. Additionally a load with the name "shiftLoad" will be loaded whenever a shift occurs.*

shiftDelayTime	0	scalar	sec	Length of time before the moving window is activated.
gasOffTime	-1.0	scalar	sec	Time at which to set the neutral gas density to zero.
Frandseed	0	int		Seed for random number generator.
B01	0.0	scalar	Tesla	Uniform static magnetic field in x1 direction.
B02	0.0	scalar	Tesla	Uniform static magnetic field in x2 direction.
B03	0.0	scalar	Tesla	Uniform static magnetic field in x3 direction.
B01analytic	"0"	string	Tesla	x1 component of the static magnetic field as a function of x1 and x2
B02analytic	"0"	string	Tesla	x2 component of the static magnetic field as a function of x1 and x2
B03analytic	"0"	string	Tesla	x3 component of the static magnetic field as a function of x1 and x2
Bf	NULL	string		Name of the file that contains the values of the magnetic field at each node.
E1init	"0"	string	V/m	x1 component of an initial electric field given as an analytic function of x1 and x2
E2init	"0"	string	V/m	x2 component of an initial electric field given as an analytic function of x1 and x2
E3init	"0"	string	V/m	x3 component of an initial electric field given as an analytic function of x1 and x2
B1init	"0"	string	Tesla	x1 component of an initial magnetic field given as an analytic function of x1 and x2
B2init	"0"	string	Tesla	x2 component of an initial magnetic field given as an analytic function of x1 and x2
B3init	"0"	string	Tesla	x3 component of an initial magnetic field given as an analytic function of x1 and x2
j1BeamDump	-1	int		Grid coordinate of left side of beam dump (magnetic field). $-1 => donotuse$
j2BeamDump	-1	int		Grid coordinate of right side of beam dump (magnetic field). $-1 => donotuse$
MarderIter	0	int		Number of iterations of Marder correction for non-charge-conserving particle current.
MarderParameter	0.5	scalar		Relaxation parameter for Marder correction.
ElectrostaticFlag	0	flag		Determines field solver: 0=electromagnetic 1-4=electrostatic: 1 = DADI 2 = Conjugate Gradient 3 = GMRES 4 = multigrid 5 = periodic DADI 6 = PETSC
initPoissonSolve	1	flag		Do an initial electrostatic Poisson solve at t=0. Default is to do it but this is seldom necessary.
CurrentWeighting	0	flag		Determines current weighting for EM model: 0=charge conserving (Nearest grid point) 1=bilinear in current (may require divergence correction).
DivergenceCleanFlag	0	flag		Determines whether a divergence correction will be applied to the electric field: 0=no, 1=yes
BoltzmannFlag	0	flag		Determines whether to use Boltzmann (massless) electrons: 0=explicit PIC electrons 1=zero electron mass
BoltzmannTemp	1	scalar	ev	Temperature of the Boltzmann electrons.
BoltzmannDensity	0	scalar	m^3	Neutral density of the Boltzmann electrons n0 from n = n0 $e^(- e * Phi/kT)$.
BoltzmannCharge	-1	scalar	1.602E-19 C	Normalized charge of the Boltzmann electrons.
BoltzmannChargeRatio	1	scalar		Charge ratio of Boltzmann source to n0.
BoltzSpecies	NULL	string		Species used in boltzmann solve.
NonRelativisticFlag	0	flag		1=NonRelativistic particle push; 0=Relativistic
nSmoothing	0	int		Number of binomial smoothing passes (0=none)

Table 9: `Species` Parameters

Parameters	Default value	Data type	Units	Description
`name`	NULL	string		Unique name for this species.
`q`	-1.60 E-19	scalar	C	Charge per physical particle of this species.
`m`	9.11 E-31	scalar	kg	Mass per physical particle of this species.
`subcycle`	1	int		Number of field advances per particle advance (> 1)
`supercycle`	1	int		Number of particle advances per field advance (> 1)
`collisionModel`	None	string		Model to use for collisions of this species: 0: none 1: electron 2: ion 3: test 4: boltzmann
`threshold`	0	scalar	eV	If the total energy of the particle drops below this value then it is transferred to the BoltzSpecies.
`particleLimit`	1e8	int		Maximum number of particles of this species in the simulation. If this value is exceeded then the number of particles is halved.
`rmsDiagnosticsFlag`	0	int		Set to 1 to collect time history plots of RMS values for beam size and velocity.
`InfiniteBFlag`	*0*	*flag*		*1=NonRelativistic, infinite B1; 0=normal, relativistic particle push*
`FieldSubFlag`	*1*	*int*		*Number of EM advances per particle advance (> 1)*
`np2cFactor`	*1*	*scalar*		*Increases the numerical weighting (np2c) of particles read in from the dump file.*
`particleLimit`	*1e8*	*scalar*		*max number of particles in the simulation. Halves the number of particles when exceeded.*
`presidue`	*1e-3*	*scalar*		*Specifies a residue for the Poisson Solver.*

Input Block Example:

```
Control
{
   dt = 3.0E-10           // time step in seconds
   ElectrostaticFlag = 1   // specifies electrostatic simulation
                           // field solver
}
```

6.2 Particle Creation

The following elements of an input file allow for the specification of initial or dynamic particle distributions as well as the types and properties of particles created in this way.

6.2.1 `Species`

The `Species` element specifies the characteristics of a macroparticle type modeled by the simulation. A macroparticle is an assemblage of np2c (to be described in the `Load` element) individual, physical particles of the same type that is modeled as a single unit. Each type of macroparticle requires its own, unique `Species` element. For simulations that model different species here will be more than one `Species` element within an input file.

Identical particle types can be created within OOPIC Pro by different sources. Electrons can be injected by a beam emitter and spawned by some interaction process such as the secondary scattering of electrons. In this situation the definition of a `Species` element for each source will track the created particles independently even though they have the same physical properties. They are treated the same by the physics kernel.

Table 10: Parameters for Monte Carlo Collisions

Parameters	Default value	Data type	Units	Description
gas	NULL	string		Background gas species. Possible values are: H, He, Li, Ar, Ne or Xe.
pressure	0	scalar	Torr	Uniform pressure for this gas.
temperature	0.025	scalar	eV	Uniform gas temperature.
eSpecies	NULL	string		Electron species to create in ionization collisions.
iSpecies	NULL	string		Ion species to be created in ionization collisions.
iSpeciesPlusPlus	NULL	string		Ion species to be created in ionization collisions.
ionzFraction_i	1	int		
ecxFactor	1	scalar		Scale all electron cross sections for this gas by ecxFactor
icxFactor	1	scalar		Scale all ion cross sections for this gas by icxFactor.
relativisticMCC	0	int		Flag: 0=Original MCC model 1=Relativistic MCC model
x1MinMKS	-1.	scalar	m	minimum x1 position defining neutral gas region
x1MaxMKS	-1.	scalar	m	maximum x1 position defining neutral gas region
x2MinMKS	-1.	scalar	m	minimum x2 position defining neutral gas region
x2MaxMKS	-1.	scalar	m	maximum x2 position defining neutral gas region
analyticF	0	string		Analytic function of x1, x2 describing the neutral gas density. The default evaluation of the function will return 0.0
collisionFlag	1	int		Enable (1) or disable (0) Monte Carlo collisions
tunnelingIonizationFlag	0	int		Enable (1) or disable (0) the tunneling ionization
ETIPolarizationFlag	0	int		Specifies E field polarization (0 = linear, 1 = circular) in the calculation of tunneling ionization probability. Requires tunneling ionization to be enabled if circular polarization is to be used.
EfieldFrequency	-1.	scalar		Frequency of the external alternating E field
TI_numMacroParticlesPerCell	10	scalar		Number of macro particles in a cell due to tunneling ionization. Requires tunneling ionization to be enabled.

Input Block Example:

```
Species
{
   name = neon
   m = 3.376e-26
   q = 1.6E-19
   subcycle = 20
   collisionModel=2
}
```

6.2.2 MCC (Monte Carlo Collisions)

This element specifies the Monte Carlo collision parameters used to model collisions of particles with background gases that might result in the ionization of the background gases and the production of a pre-defined species of particle.

Input Block Example:

```
MCC
```

```
{
   gas = Ne                // name of the neutral gas
   pressure = 0.045        // pressure of the neutral gas
   eSpecies = electrons    // electron species produced by collisions
   iSpecies = neon         // ion species produced by collisions
}
```

6.2.3 Load

These parameters specify the initial spatial and Maxwellian velocity distributions of each particle species that has been defined.

Table 11: Load Parameters

Parameters	*Default value*	*Data type*	*Units*	*Description*
name	*"noname"*			*Name of this load*
speciesName	*NULL*	*string*		*Refers to the species with the same speciesName.*
units	*MKS*	*string*	*MKS or EV*	*Determines units used*
v1drift	*0*	*scalar*	*m/s or eV*	*Drift velocity in x1 direction*
v2drift	*0*	*scalar*	*m/s or eV*	*Drift velocity in x2 direction*
v3drift	*0*	*scalar*	*m/s or eV*	*Drift velocity in x3 direction*
temperature	*0*	*scalar*	*m/s or eV*	*Isotropic temperature (3/2 kT = 1/2 m (V[tx]2+V[ty]2+V[tz]2) = 1/2 m V[t]2).*
v1thermal	*0*	*scalar*	*m/s or eV*	*Thermal velocity in x1. (1/2 kT = 1/2 m V[tx]2)*
v2thermal	*0*	*scalar*	*m/s or eV*	*Thermal velocity in x2. (1/2 kT = 1/2 m V[ty]2)*
v3thermal	*0*	*scalar*	*m/s or eV*	*Thermal velocity in x3. (1/2 kT = 1/2 m V[ty]2)*
Ucutoff	*0*	*scalar*	*m/s or eV*	*Isotropic upper temperature cutoff*
v1Ucutoff	*0*	*scalar*	*m/s or eV*	*Upper cutoff velocity for x1*
v2Ucutoff	*0*	*scalar*	*m/s or eV*	*Upper cutoff velocity for x2*
v3Ucutoff	*0*	*scalar*	*m/s or eV*	*Upper cutoff velocity for x3*
Lcutoff	*0*	*scalar*	*m/s or eV*	*Isotropic lower temperature cutoff*
v1Lcutoff	*0*	*scalar*	*m/s or eV*	*Lower cutoff velocity for x1*
v2Lcutoff	*0*	*scalar*	*m/s or eV*	*Lower cutoff velocity for x2*
v3Lcutoff	*0*	*scalar*	*m/s or eV*	*Lower cutoff velocity for x3*
density	*0*	*scalar*	*m^3*	*Uniform number density.*
x1MinMKS	*0*	*scalar*	*m*	*Lower x1 coordinate of region.*
x1MaxMKS	*0*	*scalar*	*m*	*Upper x1 coordinate of region.*
x2MinMKS	*0*	*scalar*	*m*	*Lower x2 coordinate of region.*
x2MaxMKS	*0*	*scalar*	*m*	*Upper x2 coordinate of region.*
np2c	*1e6*	*scalar*		*Number of physical particles to computer particles. If np2c = 0; unmoving background rho.*
analyticF	*0*	*string*		*Spatial distribution of particles to be loaded expressed as function of x1 and/or x2.*
LoadMethodFlag	*"Random"*	*int*		*Loads using random numbers if 0; bit reversed if 1.*

Input Block Example:

```
Load
{
   name = thisload         // does not have to be name of a species
   x1MinMKS = 0.00
   x1MaxMKS = 0.05
   x2MinMKS = 0.00
   x2MaxMKS = 0.01
```

```
   speciesName = electrons // this must correspond to a
                           // name in a Species element
   density = 1.0e14
   np2c = 1e5
   temperature = 5.93e5
}
```

6.2.4 VarWeightLoad

VarWeightLoad provides the same parameters as Load but adjusts their values to keep the macroparticle density uniform in cylindrical space by linearly increasing the weighting factor np2c with the radius.

6.2.5 PlasmaSource

A plasma source is a rectangular region in which plasma is generated at a given rate. The plasma in this case can consist of two species of particles that have been defined in a Species element. Each species in the plasma can be assigned its own properties regarding velocity, temperature, etc. Use multiple PlasmaSource elements if more than two species are needed. If the analyticF is the same then all species will originate within the same time and volume.

Table 12: PlasmaSource Parameters

Parameters	*Default value*	*Data type*	*Units*	*Description*
SpeciesName1	*NULL*	*string*		*A species defined in a Species element*
units1	*MKS*	*string*	*MKS or eV*	*Units for ALL the velocities for species 1*
v1drift1	*0.0*	*scalar*	*m/s or eV*	*Drift velocity for x1.*
v2drift1	*0.0*	*scalar*	*m/s or eV*	*Drift velocity for x2.*
v3drift1	*0.0*	*scalar*	*m/s or eV*	*Drift velocity for x3.*
temperature1	*0.0*	*scalar*	*m/s or eV*	*Isotropic temperature (see Load)*
v1thermal1	*0.0*	*scalar*	*m/s or eV*	*Thermal velocity in x1 (see Load)*
v2thermal1	*0.0*	*scalar*	*m/s or eV*	*Thermal velocity in x2 (see Load)*
v3thermal1	*0.0*	*scalar*	*m/s or eV*	*Thermal velocity in x3 (see Load)*
Ucutoff1	*0.0*	*scalar*	*m/s or eV*	*isotropic upper cutoff for temperature.*
v1Ucutoff1	*0.0*	*scalar*	*m/s or eV*	*upper cutoff velocity in x1.*
v2Ucutoff1	*0.0*	*scalar*	*m/s or eV*	*upper cutoff velocity in x2.*
v3Ucutoff1	*0.0*	*scalar*	*m/s or eV*	*upper cutoff velocity in x3.*
Lcutoff1	*0.0*	*scalar*	*m/s or eV*	*isotropic lower cutoff for temperature.*
v1Lcutoff1	*0.0*	*scalar*	*m/s or eV*	*lower cutoff velocity in x1.*
v2Lcutoff1	*0.0*	*scalar*	*m/s or eV*	*lower cutoff velocity in x2.*
v3Lcutoff1	*0.0*	*scalar*	*m/s or eV*	*lower cutoff velocity in x3.*
SpeciesName2	*NULL*	*string*		*A species defined in a Species element*
units2	*MKS*	*string*	*MKS or eV*	*Units for ALL the velocities for species 1*
v1drift2	*0.0*	*scalar*	*m/s or eV*	*Drift velocity for x1.*
v2drift2	*0.0*	*scalar*	*m/s or eV*	*Drift velocity for x2.*
v3drift2	*0.0*	*scalar*	*m/s or eV*	*Drift velocity for x3.*
temperature2	*0.0*	*scalar*	*m/s or eV*	*Isotropic temperature (see Load)*
v1thermal2	*0.0*	*scalar*	*m/s or eV*	*Thermal velocity in x1 (see Load)*
v2thermal2	*0.0*	*scalar*	*m/s or eV*	*Thermal velocity in x2 (see Load)*
v3thermal2	*0.0*	*scalar*	*m/s or eV*	*Thermal velocity in x3 (see Load)*
Ucutoff2	*0.0*	*scalar*	*m/s or eV*	*isotropic upper cutoff for temperature.*
v1Ucutoff2	*0.0*	*scalar*	*m/s or eV*	*upper cutoff velocity in x1.*
v2Ucutoff2	*0.0*	*scalar*	*m/s or eV*	*upper cutoff velocity in x2.*
v3Ucutoff2	*0.0*	*scalar*	*m/s or eV*	*upper cutoff velocity in x3.*
Lcutoff2	*0.0*	*scalar*	*m/s or eV*	*isotropic lower cutoff for temperature.*

v1Lcutoff2	*0.0*	*scalar*	*m/s or eV*	*lower cutoff velocity in x1.*
v2Lcutoff2	*0.0*	*scalar*	*m/s or eV*	*lower cutoff velocity in x2.*
v3Lcutoff2	*0.0*	*scalar*	*m/s or eV*	*lower cutoff velocity in x3.*
sourceRate	*0.0*	*scalar*	*#/m3/s*	*rate of plasma generation m-3 s-1*
np2c	*1.e6*	*scalar*		*ratio of physical to computer particles*
analyticF	*"0.0"*	*string*		*Spatial distribution of the generated particles. Understands the variables x1, x2, and t.*

6.3 Boundary Conditions

This collection of elements included within the `Region` *block supports the definition of a number of physical boundary configurations that can be encountered in plasma modeling. Common to all possible configurations, there is a generic set of parameters that is used to specify geometric and time-dependent properties of the boundaries. These generic parameters are discussed first because they are incorporated (through OOP software inheritance) with other elements that describe material properties. When material properties of a boundary segment are defined in a specific C++ class, geometric and time-dependent properties for the boundary segment are 'inherited' from a higher-level class. The inheritance of parameters is noted in later sections by stating which sets of parameters are included in the definition.*

6.3.1 Generic Boundary Conditions (Boundary Geometry)

These parameters are used to define the endpoints of a line. The orientation of the material bounded by this line segment is determined by the value of the normal parameter (see below).

Table 13:

Parameters	Default value	Data type	Units	Description
`j1`	-1	Int	Grid spacing	x1 index for first boundary endpoint.
`k1`	-1	Int	Grid spacing	x2 index for first boundary endpoint.
`j2`	-1	Int	Grid spacing	x1 index for second boundary endpoint.
`k2`	-1	Int	Grid spacing	x2 index for second boundary endpoint.

Alternatively, endpoints may be expressed in MKS units. However, OOPIC Pro will shift the computational boundary to coincide with the nearest grid point. The computational grid is specified in the `Grid` *element.*

Table 14:

Parameters	Default value	Data type	Units	Description
`A1`	-1	scalar	M	x1 location for first boundary endpoint.
`A2`	-1	scalar	M	x2 location for first boundary endpoint.
`B1`	-1	scalar	M	x1 location for second boundary endpoint.
`B2`	-1	scalar	M	x2 location for second boundary endpoint.

`Segment` *Boundary endpoints can also be specified by using the* `Segment` *specifier within the definition of a boundary type if the material all along the multi-segmented boundary is the same. This aids the definition of boundary condition geometries of a complex nature that require the definition of multiple, connected line segments to specify the material boundary. It is also an aid to efficient computation. Complex geometries are defined by multiple* `Segment` *specifiers. Not all boundary types can utilize multiple* `Segment` *specifiers. This situation is noted in the explanation of each boundary type that follows.*

Input Block Example:

```
<BoundaryGroup>
{
   //             generic and material parameters
  Segment
  {
        SegName = element1
        A1 = 0
        B1 = 0.3
        A2 = 0
        B2 = 0
        normal = 1
  }
  Segment

  {
        SegName = element2
        A1 = 0.3
        B1 = 0.3
        A2 = 0
        B2 = 0.3
        normal = -1
  }
   //             other generic and material parameters
}
```

An example of a complex, multi-segment boundary is provided in the example specification for an `Equipotential` *boundary condition below. Not all boundary condition types will support multiple segments.*

6.3.2 Generic Boundary Conditions (Boundary Material Properties)

Table 15: Generic Boundary Conditions (Boundary Material Properties)

Parameter	*Default value*	*Data type*	*Units*	*Description*
`name`	*"Noname"*	*string*		*A name for this boundary.*
`normal`	*1*	*int*		*Unit direction of the face normal: -1=down /left, 1=up/right. For oblique segments the unit direction of the normal is: 1 for right face of boundary, -1 for left face of boundary*
`fill`	*0*	*int*		*If 1, the boundary is filled in. This implies a closed boundary.*
`EFFlag`	*0*	*flag*		*Data accumulator for Poynting flux through boundary: 0=off; 1=on.*
`IdiagFlag`	*0*	*flag*		*Data accumulator for particle current: 0=off; 1=on.*
`Ihist_avg`	*1*	*int*		*Number of time steps for averaging current plots.*
`Ihist_len`	*1024*	*int*		*Length of the current history arrays.*
`nxbins`	*0*	*int*		*Number of spatial bins along boundary segment for distribution accumulation.*
`nenergybins`	*0*	*int*		*Number of energy bins for distribution accumulation.*
`nthetabins`	*0*	*int*		*Number of bins for angular distribution function.*
`theta_min`	*0*	*scalar*		*Minimum angle for angular distribution function.*
`theta_max`	*90*	*scalar*		*Maximum angle for angular distribution function.*
`energy_min`	*0*	*scalar*	*eV*	*Minimum energy for dist. function collector.*
`energy_max`	*100*	*scalar*	*eV*	*Minimum energy for dist. function collector.*
`C`	*1.0*	*scalar*		*DC value for time-dependent function.*

A	0.0	scalar		AC value for time-dependent function.
frequency	0.0	scalar	Hz	Frequency of AC variation for time-dependent function.
phase	0.0	scalar	rad	Initial phase of AC variation for time-dependent function.
tdelay	0.0	scalar	s	Time to delay time before envelope function starts.
trise	0.0	scalar	s	Rise time of envelope for time-dependent function.
tpulse	10000.0	scalar	s	Pulse width of envelope for time-dependent function.
tfall	0.0	scalar	s	Fall time of envelope for time-dependent function.
a0	1.0	scalar		Base value of envelope for time dependent function.
a1	1.0	scalar		Maximum value of envelope for time-dependent function.
xtFlag	0	flag		Selects space or time dependence of boundary condition: 0 = envelope * sinusoid 1 = f(t) 2 = f(x) 3 = f1(x)*f2(t) 4 = f(x;t)
F	"100"	string		A string which is interpreted at runtime to give the space and/or time-dependence of this boundary

6.3.3 Dielectric

To specify a boundary composed of a dielectric material all generic boundary parameters, as given above, are utilized as well as the following parameters for the specification of the material properties of the boundary. Only one `Segment` *specifier within the* `Dielectric` *element is permitted for the definition of the physical boundary.*

Table 16: `Dielectric` Parameters

Parameters	Default value	Data type	Units	Description
er	1.0	scalar		Relative permittivity
QuseFlag	1	Flag		Determines whether to use the accumulated surface charge in the field solve; i.e. 0 indicates that the charge drains off. This flag is only applicable to the electrostatic model.
reflection	0	scalar		Coefficient for particle reflection. Value is between 0 and 1.
refMaxE	1.e10	scalar	eV	Only reflect particles below this energy. `Secondary` and `Secondary2` groups can contain zero or more of these.

Input Block Example (from dc-NeXe.inp):

```
Dielectric
{
   er = 1.0
   j1 = 0
   j2 = 30
   k1 = 30
   k2 = 30
   name = bulb
   normal = -1
}
```

6.3.4 `Conductor`

Specification of a boundary with conducting properties is accomplished by utilizing any or all of the Generic parameters above in conjunction with those included in the `Dielectric` *element. Multiple* `Segment` *specifiers are permitted.*

Input Block Example:

```
Conductor
{
   j1 = 0
   k1 = 10
   j2 = 0
   k2 = 50
   normal = 1
}
```

6.3.5 `Equipotential`

Specification of an `Equipotential` *boundary incorporates the same set of parameters defined for a* `Conductor` *boundary segment.*

Note that an `Equipotential` *boundary will use the time-dependent function parameters in the Generic boundary parameter set to define a spatially uniform time-dependent voltage on a surface. This functions as a grounded* `Conductor` *in the electromagnetic model.*

If multiple boundary segments, all at the same potential, are part of the same equipotential surface then it is important for performance that they are all specified with same `Equipotential` *element. The example below includes two* `Segment` *specifiers within the same* `Equipotential` *element. Computational efficiency makes this preferable to defining solitary* `Segment` *specifiers within two* `Equipotential` *elements.*

Input Block Example (from hollow.inp):

```
Equipotential
{
  xtflag = 0         // using envelope * sinusoid
  C = -2000          // DC offset
  A = 500            // Amplitude of AC component
  frequency = 1E8    // AC frequency
  phase = 90         // AC phase
  trise = 0          // envelope rise time
  tpulse = 15e-6     // duration of nonzero envelope
  tfall = 1e-9       // fall of envelope
  a1 = 1             // max scale of envelope
  a0 = 0             // min scale of envelope
  Secondary          // secondary electrons emitted upon impact
  {                  // of argon ions with 30\% probability
    secondary = 0.3
    secSpecies = electrons
    iSpecies = argon
  }
  Segment
  {                  // left boundary of equipotential surface
   j1 = 0
   k1 = 0
   j2 = 0
   k2 = Kmax
   normal = 1
```

```
  }
  Segment
  {                  // top boundary of equipotential surface
   j1 = 0
   k1 = Kmax
   j2 = jRightSideWall
   k2 = Kmax
   normal = -1
  }
}
```

6.3.6 `Polarizer`

Specification of a `Polarizer` *boundary incorporates the same set of parameters defined for a* `Conductor` *boundary segment.*

Table 18: `Secondary` Element Parameters

Parameters	Default value	Data type	Units	Description
`secondary`	0	scalar		Secondary emission coefficient; can be > 1
`threshold`	0.5	scalar	eV	Emission threshold incident energy must exceed for emission to occur
`Eemit`	2	scalar	eV	Maximum energy of emitted secondary particles (uniformly distributed).
`secSpecies`	NULL	string		Name of the species to emit as secondary particles; must correspond to an existing species if secondary $>$ 0.
`iSpecies`	NULL	string		Incident species to generate as secondary particles

Table 17: `Polarizer` Parameters

Parameters	Default value	Data type	Units	Description
`transmissivity`	0	scalar		Fraction (between 0 and 1) of particles which pass through the polarizer
`Phi`	90	String	degrees	Polarization angle with respect to the simulation plane. This parameter can be a function of distance along the polarizer boundary as well. X = 0 is defined as the lower- or left-most point of the polarizer, and X = 1 is the other end.

Input Block Example:

```
Polarizer
{
 name = Polarizer
 A1 = 0
 A2 = Rdrift
 B1 = Zmax
 B2 = Rdrift
 normal = -1
 Phi = 45*pulse(X,Ramp1Start,Ramp1Length,HelixLength,Ramp2Length)
}
```

6.3.7 `Secondary`

A `Secondary` element can be nested within any boundary element specification to include the production of a secondary species. Production of the secondary species modeled in OOPPIC Pro employs a phenomenological model of the quantum processes that occur in the surface of materials.

For an example of `Secondary` usage, see `Equipotential` above. The `Secondary` element implements simple step function with a turn-on threshold to model secondary emission.

6.3.8 `Secondary2`

The `Secondary2` element implements a Vaughan-based secondary production model that includes energy and angular dependence as well as a full emission spectrum including reflected and scattered primaries. This is a more accurate model of secondary particle production.

The coefficient of secondary emission is the ratio of the numbers of ejected to incident electrons. It is given by:

$$\delta(\epsilon,\theta) = \delta_{max\,0}(1 + k_{s\delta}\frac{\theta^2}{2\pi})(w\,exp(1-w))^k$$

where ϵ is the incident energy of a particle and θ is its angle of incidence measured with respect to the surface normal. $k_{s\delta}$ is a surface smoothness parameter (described below), δ_{max0} is the peak secondary emission coefficient corresponding to the energy $\epsilon_{max\,0}$and normal incidence. The exponent k is

$$k = \begin{cases} 0.62,\ w < 1 \\ 0.25,\ w \geq 1 \end{cases}$$

Values for k are derived from a curve-fit analysis.

w is the normalized energy given by:

$$w = \frac{\epsilon - \epsilon_0}{\epsilon_{max\,0}(1 + k_{sw}\theta^2/2\pi) - \epsilon_0}$$

where ϵ_0 is the secondary emission threshold (as defined for the `Secondary` element above). k_{sw} is a surface-smoothness parameter similar to $k_{s\delta}$ (both can vary between 0 for rough surfaces and 2 for polished surfaces).

The nature of particle interaction is determined by comparing a uniform random variable, R, whose values are between 0 and 1 to (fReflected + fScattered). A true secondary particle is generated from a Maxwellian distribution if (fScattered + fReflected) $<$ R. Otherwise a single component of the incident particle's velocity has its sign reversed. Orientation of the reflecting surface determines which component is changed. Further, if R $>$ fReflected, all components of the particle's velocity are scaled by uniform (0,1) random values.

Particle interaction obeys the rule fScattered + fReflected + fSecondary2 = 1.

`Secondary2` utilizes the parameters defined for `Secondary`.

Table 19: Additional Parameters for `Secondary2`

Parameters	*Default value*	*Data type*	*Units*	*Description*
`fReflectedi`	*0*	*scalar*		*Fraction of emitted particles that are reflected primaries.*
`fScattered`	*0*	*scalar*		*Fraction of emitted particles that are scattered primaries.*
`energy_max0`	*0*	*scalar*		*eV Impact energy at maximum particle yield (e[max0]).*
`ks`	*1*	*scalar*		*Surface roughness; 0 <= ks <= 2 (0 is rough, 2 is smooth)*

6.3.9 `DielectricRegion`

Specification of a `DielectricRegion` boundary incorporates the same set of parameters defined for a `Dielectric` boundary segment. Only one `Segment` specifier is permitted.

For `DielectricRegion`, the points formed by (j1,k1) and (j2,k2) indicate opposite corners of a rectangular region separate from the model boundaries rather than a line segment as would be used for a boundary. This provides the means to define a body with dielectric properties completely contained within the modeled system but detached from its boundaries.

Table 20: Current Region Parameters

Parameters	Default value	Data type	Units	Description
`Current`	1	scalar	Amp	Magnitude of the TOTAL current in the region.
`direction`	3	int		Direction of current: x1, x2 or x3
`currentFile`	NULL	string		File containing current distribution
`analyticF`	0.0	string		Analytic function of x1, x2 describing the current distribution

Input Block Example (from dring.inp):

```
DielectricRegion
{
        er = 100.0
        j1 = 3*Jmax/13
        k1 = 3*Kmax/13
        j2 = 10*Jmax/13
        k2 = 10*Kmax/13
        QuseFlag = 1
}
```

6.3.10 `DielectricTriangle` (no example files)

Specification of a `DielectricTriangle` boundary incorporates the same set of parameters defined for a `Dielectric` boundary segment. Only one `Segment` specifier is permitted.

Similar to `DielectricRegion`, this element allows a body to be defined within the model boundaries but with a triangular, rather than rectangular, shape. For `DielectricTriangle`, the points formed by (j1,k1) and (j2,k2) indicate the two vertices of the base of a right triangle. The third vertex is determined by the parameter normal. If normal > 0, the third vertex of the triangle is above the line. Otherwise, it will be below the line.

6.3.11 `CurrentRegion`

Specification of a `CurrentRegion` boundary incorporates the Generic set of parameters plus the parameters in the table. Only one `Segment` specifier is permitted.

Input Block Example (from klystrino.inp):

```
CurrentRegion
{
   name = InputSig
   analyticF = 1
   A1 = L * 0.5 + dZ
   A2 = Ro - 2 * dR
   B1 = L * 0.5 -dZ + GapLength
   B2 = Ro - 3 * dR
   C = 0
   A = 1000
   direction = 1
   frequency = 2 * 3.1415926 * Freq
   a0 = 0
   a1 = 1
   tdelay = 0
   trise = 0
   tpulse = 100000 * DT
   tfall = 0
```

```
}
```

6.3.12 `Iloop`

Table 21: `Iloop` Parameters

Parameters	Default value	Data type	Units	Description
`Current`	1	scalar	Amp	Magnitude of the current in loop.

Input Block Example (from cavity.inp):

```
Iloop
{
   j1 = 0
   j2 = 20
   k1 = 0
   k2 = 10
   Current = 1
   frequency = 1.5e9
   C = 0
   A = 1
   tpulse = 3.33e-10
   a0 = 0
   trise = 0
   tfall = 0
}
```

6.3.13 `CylindricalAxis`

Specification of a `CylindricalAxis` boundary incorporates the Generic set of parameters. This element is necessary only in cylindrical coordinates if r=0 is included in the region.

Input Block Example:

```
CylindricalAxis
{
   j1 = 0
   k1 = 0         // k1 and k2 have to be zero, since boundary is
                  // on the x1 axis
   j2 = Jmax
   k2 = 0
   normal = 1
}
```

6.4 Ports

This collection of elements included within the `Region` block supports the definition of a number of free-space (void of physical material) boundary configurations that can be encountered in plasma modeling. As with the specification of physical boundaries there is an inheritance of, or inclusion of, the generic boundary parameters outlined earlier in Section 6.3 of this manual.

6.4.1 ExitPort

An `ExitPort` *is used when an EM wave or a particle is to be treated as if it were escaping the defined boundaries of the simulation.*

Specification of a `ExitPort` *boundary incorporates the Generic set of Boundary Condition parameters plus the parameters in the table. Only one* `Segment` *specifier is permitted.*

Table 22: `ExitPort` Parameters

Parameters	Default value	Data type	Units	Description
`R`	376.99	scalar	ohms	Resistive component of surface impedance.
`L`	0	scalar	H	Inductive component of surface impedance.
`Cap`	0	scalar	F	Capacitive component of surface impedance.
`Rin`	376.99	scalar	ohms	Resistive component of surface impedance for incoming wave.
`Lin`	0	scalar	H	Inductive component of surface impedance for incoming wave.
`Capin`	0	scalar	F	Capacitive component of surface impedance for incoming wave.

Input Block Example:

```
ExitPort
{
   j1 = 0
   k2 = 0
   j2 = 0
   k1 = 0.04
   normal = 1
   EFFlag = 1
   name = below beam
   C = 0
   A = 0
   frequency = 1
}
```

6.4.2 Gap

Specification of a `Gap` *boundary incorporates the Generic set of parameters. Only one* `Segment` *specifier is permitted. Note that a* `Gap` *boundary will use the time-dependent function parameters in the Generic boundary parameter set to define a spatially uniform time-dependent voltage on a surface. This works to set a time-dependent field (V/m) uniformly in space across the gap.*

Input Block Example:

```
Gap
{
  j1 = cav1gL/DeltaZ
  k2 = Rmax/DeltaR
  j2 = cav1gStumpL/DeltaZ
  k1 = Rmax/DeltaR
  normal = -1
  A = -3.2e6
  frequency = 1.34e9
  phase = 0
  EFFlag = 0
  name = Gap Left
}
```

Table 23: `PortGauss` Parameters

Parameters	Default value	Data type	Units	Description
`waveLeng_p0`		scalar	M	wavelength of the EM wave
`amp_p0`		scalar	V/m	peak wave amplitude of electric field
`spotSize_p0`		scalar	M	The beam half width at the waist.
`focus_p0`		scalar	M	Displacement of the focus.
`chirp_p0`		scalar		
`pulShp_p0`		int		pulse shape: 0=trapezoid 1=Gaussian 2=half-sine 3=wide Gaussian
`pulLeng_p0`		scalar	M	Pulse length: half-width for Gaussian, full for half-sine and trapezoid
`tdelay_p0`		scalar	S	time delay to start pulse.
`offset`	0	scalar	m	offset from the center of the simulation to the peak of the Gaussian

6.4.3 `PortGauss`

`PortGauss` launches two linearly polarized electromagnetic pulses in 2-D Cartesian geometry. The pulses are launched from a given boundary by controlling the temporal and spatial dependence of the electric field at that boundary. The transverse spatial profile is Gaussian along the boundary. The temporal variation can be either trapezoidal, Gaussian or a half-sine. Pulse 0 is linearly polarized in the y direction, while pulse 1 is linearly polarized in the z direction. The pulses are launched in accordance with paraxial theory, such that the waist (focus) is a given distance from the port.

The parameters of pulse 0 incorporate the Generic set of parameters. Only one `Segment` specifier is permitted

The parameters of pulse 1 (linearly polarized in the z direction) are the same, except that "`_p0`" is replaced by "`_p1`". To have only one pulse, set the amplitude of the other to zero.

*OOPIC Pro will abort if the value of focus is set to zero. Set the value of the focus equal to one or two cell lengths inside the region (Dx or 2*Dx) if the focus needs to be at the boundary.*

For `pulShp_p0` = 1 (gaussian shape), at t = tdelay +/- tpulse/2, the value is $exp(-1)$, and zero beyond that. For `pulShp_p0` = 3 (wide gaussian shape), the pulse is actually cut off further out (meaning the pulse is 6x longer), so the minimum value is 4.8e-6 with respect to the maximum.

Input Block Example: (from loasisHe.inp)

```
PortGauss
{
  j1 = 0
  k1 = 0
  j2 = 0
  k2 = Ny
  normal = 1

  A = 0
  C = 1.0

// Wave (0)
  pulShp_p0 = nPulseShape
  tdelay_p0 = 0.0
  pulLeng_p0 = pulseLength
  chirp_p0 = 0.0
  spotSize_p0 = waistSize
  waveLeng_p0 = laserWavelength
  focus_p0 = waistLocation
  amp_p0 = 0.0

// Wave (1)
  pulShp_p1 = nPulseShape
```

```
  tdelay_p1 = 0.0
  pulLeng_p1 = pulseLength
  chirp_p1 = 0.0
  spotSize_p1 = waistSize
  waveLeng_p1 = laserWavelength
  focus_p1 = waistLocation
  amp_p1 = peakElectricField

  EFFlag = 0

  name = PortGauss
}
```

6.5 Particle Emitters

This set of elements are used to specify a variety of ways in which particles may be created in the modeled system.

6.5.1 Generic Emitter Parameters

Specification of generic parameters for particle emitters incorporates the specification of Dielectric Boundary Condition parameters. Only one `Segment` *specifier is permitted.*

This set of parameters is very similar to that used for `Load`*.*

Table 24: Generic Emitter Parameters

Parameters	*Default value*	*Data type*	*Units*	*Description*
`units`	*MKS*	*string*	*MKS or EV*	*units for ALL the velocities*
`v1drift`	*0*	*scalar*	*m/s or eV*	*Drift velocity in x1.*
`v2drift`	*0*	*scalar*	*m/s or eV*	*Drift velocity in x2.*
`v3drift`	*0*	*scalar*	*m/s or eV*	*Drift velocity in x3.*
`temperature`	*0*	*scalar*	*m/s or eV*	*Isotropic temperature.*
`v1thermal`	*0*	*scalar*	*m/s or eV*	*Thermal velocity in x1.*
`v2thermal`	*0*	*scalar*	*m/s or eV*	*Thermal velocity in x2.*
`v3thermal`	*0*	*scalar*	*m/s or eV*	*Thermal velocity in x3.*
`Lcutoff`	*0*	*scalar*	*m/s or eV*	*Isotropic lower cutoff for temperature.*
`v1Lcutoff`	*0*	*scalar*	*m/s or eV*	*Lower cutoff velocity in x1.*
`v2Lcutoff`	*0*	*scalar*	*m/s or eV*	*Lower cutoff velocity in x2.*
`v3Lcutoff`	*0*	*scalar*	*m/s or eV*	*Lower cutoff velocity in x3.*
`Ucutoff`	*0*	*scalar*	*m/s or eV*	*Isotropic upper cutoff for temperature.*
`v1Ucutoff`	*0*	*scalar*	*m/s or eV*	*Upper cutoff velocity in x1.*
`v2Ucutoff`	*0*	*scalar*	*m/s or eV*	*Upper cutoff velocity in x2.*
`v3Ucutoff`	*0*	*scalar*	*m/s or eV*	*Upper cutoff velocity in x3.*
`np2c`	*1.e6*	*scalar*		*Number of physical particles to computer particles.*
`speciesName`	*NULL*	*string*		*Refers to the Species with the same speciesName.*

6.5.2 `BeamEmitter`

Specification of `BeamEmitter` *boundary incorporates the Generic Emitter set of parameters. Only one* `Segment` *specifier is permitted.*

Table 25: BeamEmitter Parameters

Parameters	Default value	Data type	Units	Description
I	1	scalar	A	Emission current.
thetadot	0	scalar	rad/s	Angular velocity of beam.
nIntervals	0	int		Number of intervals for inversion of spatial dependence of J(x) as specified by F. Zero means use the number of cells.

Input Block Example:

```
BeamEmitter
{
   j1 = 0
   k1 = 0
   j2 = 0
   k2 = 10*Kmax/13
   normal = 1
   speciesName = electrons
   thetadot = 0.0
   I = 10
   np2c = 1.0E7
   v1drift = 1e8
   v2drift = 0
   v3drift = 0
   v1thermal = 0
   v2thermal = 0
   v3thermal = 0
}
```

6.5.3 `EmitPort`

`EmitPort` is a subclass of `BeamEmitter`. It applies dielectric boundary conditions to the port segment rather than conductor boundary conditions.

Input Block Example: (from voltest.inp)

```
EmitPort
{
   j1 = 9
   j2 = 11
   k1 = 20
   k2 = 20
   speciesName = electrons
   normal = -1
   np2c = 4e3     // number of electrons per macroparticle
   I = 0.000005   //current
   v2drift = 0.01
}
```

6.5.4 `VarWeightBeamEmitter`

Specification of VarWeightEmitter boundary incorporates the `BeamEmitter` (see Section 6.5.1) set of parameters. Only one `Segment` specifier is permitted. This segment is used in cylindrical coordinates. It increases the np2c weight linearly to maintain a uniform beam density.

Table 26: `VarWeightBeamEmitter` Parameters

Parameter	Default value	Data type	Units	Description
nEmit	0	int		Number of particles per time step to emit; if nEmit=0 then the number is computed using np2c.

Table 27: `FowlerNordheimEmitter` Parameters

Parameters	Default value	Data type	Units	Description
`A_FN`	1.5414e-6	scalar		Coefficient A of the Fowler-Nordheim field-emission model
`beta_FN`	1.0	scalar		Coefficient beta of the Fowler-Nordheim field-emission model
`B_FN`	6.8308e9	scalar		Coefficient B of the Fowler-Nordheim field-emission model
`C_v_FN`	0.0	scalar		Coefficient C_v of the Fowler-Nordheim field-emission model
`C_y_FN`	3.79e-5	scalar		Coefficient C_y of the Fowler-Nordheim field-emission model
`Phi_w_FN`	4	scalar	eV	Coefficient Phi_w of the Fowler-Nordheim field-emission model
`nIntervals`	0	scalar		Number of intervals to be used for emitting particles. nIntervals will be reset to the number of cells along the emitting boundary (with a minimum of 2)

6.5.5 `FowlerNordheimEmitter`

This element implements the Fowler-Nordheim law for field emission according to the following equation:

$$J_{FN} = \frac{A_{FN}(\beta_{FN}E)^2}{\Phi_w} exp\left(-\frac{B_{FN}\, v(y)\Phi_w^{3/2}}{\beta_{FN}E}\right)$$

where J_{FN} is the emitted current density in A/m^2, E is the perpendicular component of the electric field in V/m, and Φ_w is the work function of the material in eV. The other parameters have the following value or functional form:

$$y = C_y\, E^{1/2}/\Phi_w$$

$$v(y) = 1 - C_v y^2$$

$$A_{FN} = 1.5414 \times 10^{-6}$$

$$B_{FN} = 6.8308 \times 10^{9}$$

An instance of this new field-emission surface is created via an input block with the name `FowlerNordheimEmitter`. A `FowlerNordheimEmitter` incorporates the Generic Emitter set of parameters.

Input Block Example:

```
FowlerNordheimEmitter
// Below, we specify all of the Fowler-Nordheim parameters that are
```

```
//   specific to this type of particle emitter, even though most of
//   them are given the default value.
//
{
  j1 = (numCellsX - numEmitterCells) / 2
  j2 = (numCellsX + numEmitterCells) / 2
  k1 = numCellsY
  k2 = numCellsY
  normal = -1

  speciesName = electrons  // name from species group above
  np2c = 5.e+6             // numerical weight of emitted particles

  // Coefficient "A" of the Fowler-Nordheim field emission model.
  // The default value is 1.5414e-06, which is specified here.
  A_FN = 1.5414e-06

  // Coefficient "beta" of the Fowler-Nordheim field emission model.
  // This simply multiplies the electric field from the simulation.
  // The default value is 1.  Here, we specify beta\_FN = 20000, which
  //    yields non-zero field emission, without having large electric
  //    fields.

  beta_FN = 20000.


  // Coefficient "B" of the Fowler-Nordheim field emission model.
  // The default value is 6.8308e+09, which is specified here.

  B_FN = 6.8308e+09

  // Coefficient "C_v" of the Fowler-Nordheim field emission model.
  // The default value is 0, which is specified here.

  C_v_FN = 0.

  // Coefficient "C_y" of the Fowler-Nordheim field emission model.
  // The default value is 3.79e-05, which is specified here.

  C_y_FN = 3.79e-05


  // The work function "Phi_w" for electrons in the surface, in eV.
  // The default value is 4 eV, which is specified here.

  Phi_w_FN = 4.

  // The number of intervals to be used for emitting particles.
  // The default value of 0, which is specified here.
  // In the default case, nIntervals will be reset to the \# of cells
  //   along the emitting boundary (with a minimum of 2), which  is
  //   the most reasonable thing to do.

  nIntervals = 0
}
```

6.6 User-Defined Diagnostics

Along with the default diagnostics available during an OOPIC Pro simulation, new diagnostics can be created which can also be plotted when running OOPIC Pro.

A new diagnostic can be specified in an element with the heading 'Diagnostic'. The value of the diagnostic is put into the parameter VarName. The point, line, or region over which the diagnostic is to be sampled must be specified.

Table 28: Diagnostic Geometry

Parameters	Default value	Data type	Units	Description
`j1`	-1	int		x1 index for first diagnostic endpoint.
`k1`	0	int		x2 index for first diagnostic endpoint.
`j2`	0	int		x1 index for second diagnostic endpoint.
`k2`	k1	int		x2 index for second diagnostic endpoint.

As with the generic boundary conditions, diagnostic boundaries can be specified in MKS units. However, OOPIC Pro will put the diagnostic on the nearest grid point.

The type of graph plotted by OOPIC Pro for the diagnostic will be determined by the diagnostic boundaries. There are three types of graphs:

6.6.1 Spatial Regions

If (j1,k1) and (j2,k2) form the corners of a rectangle then the graph is X versus Y or R versus Z. This is good for plotting the time average of variable.

6.6.2 Time History of a Line

If (j1,k1) and (j2,k2) form the ends of a line, then the graph is a time history of the variable versus position along the line.

6.6.3 Time History of a Point

If (j1,k1) and (j2,k2) are the same point then the graph is a time history of the variable at that point.

Other parameters:

The time histories are of length HistMax. When the array is full and Comb is defined then the array is combed to every Comb'th value. If Comb is 0 then the array is a moving window. If Ave is set to a nonzero value, Ave data points are averaged together.

Currently the following diagnostic variables can be plotted (default specification). Particle positions vs. time are displayed without having to be requested.

For time histories:

Table 29: Diagnostic Parameters

Parameters	Default value	Data type	Units	Description
`A1`	0	scalar	m	x1 location for first diagnostic endpoint.
`A2`	0	scalar	m	x2 location for first diagnostic endpoint.
`B1`	0	scalar	m	x1 location for second diagnostic endpoint.
`B2`	0	scalar	m	x2 location for second diagnostic endpoint.

Table 30: Time History of a Point Parameters

Parameters	Default value	Data type	Units	Description
VarName	'NULL'	string		Name of the variable chosen from list (below) to be plotted.
x1_Label	x1	string		x1 Label of the plot
x2_Label	x2	string		x2 Label of the plot
x3_Label	x3	string		x3 Label of the plot
title	'not_named'	string		Title of window.
HistMax	64	int		maximum length of history array
save	0	flag		1: save the diagnostic data in the dumpfile. 0: restart when restarted from a dumpfile
Comb	0	int		Every Comb'th value is left when HistMax is reached.
Ave	0	int		Averaged over Ave data points when adding to history array
integral	NULL	string		one of: line (variable dotted into dl); flux (variable dotted into dS); sum (simple summation)
polarizationEB	"EyBz"	string		"EyBz" computes diagnostic for (Ey - c*Bz)/2 and (Ey + c*Bz)/2. "EzBy" specifies the calculation of (Ez-c*By)/2 and (Ez + c*By)/2 instead.
psd1dFlag	0	int		"psd1dFlag = 1" calculates the 1d power spectral densities for the two linear combinations of E and B selected via the "polarizationEB". The 1d PSD are calculated along the x axis for each value of the y index.
windowName	'Blackman'	string		Window to apply to data before the FFT. The following windows are implemented: "Blackman", "Bartlett", "Hamming", "Hann" and "Welch". (UNIX only)
nfft	0	scalar		Number of points to be used in fft computation (UNIX only)

Table 31: Time History Diagnostics

Diagnostic Name	*Content*
`E1`	*Ez (RZ) or Ex (XY)*
`E2`	*Er (RZ) or Ey (XY)*
`E3`	*Ephi (RZ) or Ez (XY)*
`B1`	*Bz (RZ) or Bx (XY)*
`B2`	*Br (RZ) or By (XY)*
`B3`	*Bphi (RZ) or Bz (XY)*
`I1`	*Iz (RZ) or Ix (XY) (only with EM field solve)*
`I2`	*Ir (RZ) or Iy (XY) (only with EM field solve)*
`I3`	*Iphi(RZ) or Iz(XY) (only with EM field solve)*
`intEdl1`	*Ez (RZ) or Ex (XY)*
`intEdl2`	*Er (RZ) or Ey (XY)*
`intEdl3`	*Ephi (RZ) or Ez (XY)*
`poynting1`	*Poynting Vector in x1 (only with EM field solve)*
`poynting2`	*Poynting Vector in x2 (only with EM field solve)*
`poynting3`	*Poynting Vector in x3 (only with EM field solve)*
`Rho`	*charge density*
`SpeciesName`	*rho of a given species*
`Phi`	*potential (only with electrostatic field solve)*
`Q`	*surface charge on dielectrics*
`LaserSpotSize`	*Integral(y*y* Ey*Ey) / Integral(Ey * Ey) over the line k1 to k2, assuming the laser spot is centered at (k2-k1)/2. Display is the average over HISTMAX time steps of this measure.*
`WaveDirDiagnostic`	*Computes (Ey - c*Bz)/2 and (Ey + c*Bz)/2 over the mesh which distinguishes left and right moving waves in the system (polarized in y).*

For spatial regions, if JdotE is the requested diagnostic then JdotE is plotted.

An example of an input file using diagnostics is: "inp/TI_H_WDD.inp".

Input Block Example:

```
Diagnostic
{
   j1 = 0         //geometry of diagnostic,
   j2 = Nx        // in this case it is a 2-D region
   k1 = 0
   k2 = Ny
   VarName = WaveDirDiagnostic
   polarizationEB = EzBy
   psd1dFlag = 1 // calculate the 1d power spectral density
   windowName = Hann
   title = Wave Energy
   x1\_Label = x
   x2\_Label = y
   x3\_Label = Wave Energy
}
```

6.7 H5Diagnostic

OOPIC Pro will selectively dump certain diagnostic data in HDF5 format. This is an important feature for OOPIC Pro runs that are performed on parallel systems where each process creates its own output file. The files are merged in a postprocessing step by utilities provided for that purpose.

Table 32: Diagnostic Geometry

Parameter	Default value	Data type	Description
j1	-1	int	x1 index for first diagnostic endpoint.
k1	0	int	x2 index for first diagnostic endpoint.
j2	0	int	x1 index for second diagnostic endpoint.
k2	k1	int	x2 index for second diagnostic endpoint.

Table 33: Diagnostic Boundaries

Parameters	Default value	Data type	Units	Description
A1	0	scalar	m	x1 location for first diagnostic endpoint.
A2	0	scalar	m	x2 location for first diagnostic endpoint.
B1	0	scalar	m	x1 location for second diagnostic endpoint.
B2	0	scalar	m	x2 location for second diagnostic endpoint.

To call for a diagnostic to be dumped in HDF5 format, specify an element with the heading 'H5Diagnostic'. The value of the diagnostic is put into the parameter VarName. The point, line, or region over which the diagnostic is to be sampled must be specified.

As with the generic boundary conditions, diagnostic boundaries can be specified in MKS units. However, OOPIC Pro will put the diagnostic on the nearest grid point.

The type of graph plotted by OOPIC Pro for the diagnostic will be determined by the diagnostic boundaries. There are three types of data collected:

6.7.1 Spatial regions

If (j1, k1) and (j2, k2) form the corners of a rectangle then the graph is X versus Y or R versus Z. This is good for plotting the time average of variable.

6.7.2 Time history of a line

If (j1, k1) and (j2, k2) form the ends of a line, then the graph is a time history of the variable versus position along the line.

6.7.3 Time history of a point

If (j1, k1) and (j2, k2) are the same point then the graph is a time history of the variable at that point.

Other parameters:

Table 34: Other Parameters

Parameters	Default value	Data type	Description
VarName	'NULL'	string	Name of the variable chosen from list (below) to be dumped.
dumpPeriod	0	int	0 => dump at same frequency as binary dump file; nonzero = number of timesteps between hdf5 dumps
filename	"Diagnostics"	string	Name of diagnostic HDF5 dump file. This will be set to ¡inputfilebase¿.h5 before first dump occurs

Table 35: Time History Diagnostics

Diagnostic Name	Content
`E1i`	Ez (RZ) or Ex (XY)
`E2`	Er (RZ) or Ey (XY)
`E3`	Ephi (RZ) or Ez (XY)
`B1`	Bz (RZ) or Bx (XY)
`B2`	Br (RZ) or By (XY)
`B3`	Bphi (RZ) or Bz (XY)
`I1`	Iz (RZ) or Ix (XY) (only with EM field solve)
`I2`	Ir (RZ) or Iy (XY) (only with EM field solve)
`I3`	Iphi(RZ) or Iz(XY) (only with EM field solve)

Currently the following diagnostic variables can be dumped if WaveDirDiagnostic and PSDFieldDiag are specified in the input file.

For time histories:

6.7.4 Limit

A Limit rule has the form:

variable op value

where variable is the name of a parameter, value is a numeric value and op is an operator from the list '==, >= , <=, > and <'. An example of a Limit rule that would restrict the simulation timestep to be nonzero and positive would be dt > 0.

6.7.5 Relation

A Relation rules has the form:

variable1 op variable2

where variable1 and variable2 are the names of two parameters (within a given ParameterGroup) and op is an operator from the same list defined for the Limit rules. This type of rule constrains two parameters to have a specified relationship. A rule requiring that the simulation timestep to satisfy certain convergence criteria would be dt <= Courant.

6.7.6 Algebra

An Algebra rule has the form:

variable1 oparith variable2 op value

*where variable1 and variable2 are the names of two parameters (within a ParameterGroup), oparith is binary arithmetic operator from the list '+. -, *, /' and op is an operator from the list given for the Limit rule.*

7 Example Problems

This section provides examples of how OOPIC Pro can be applied to the modeling of a variety of different physical devices and phenomena.

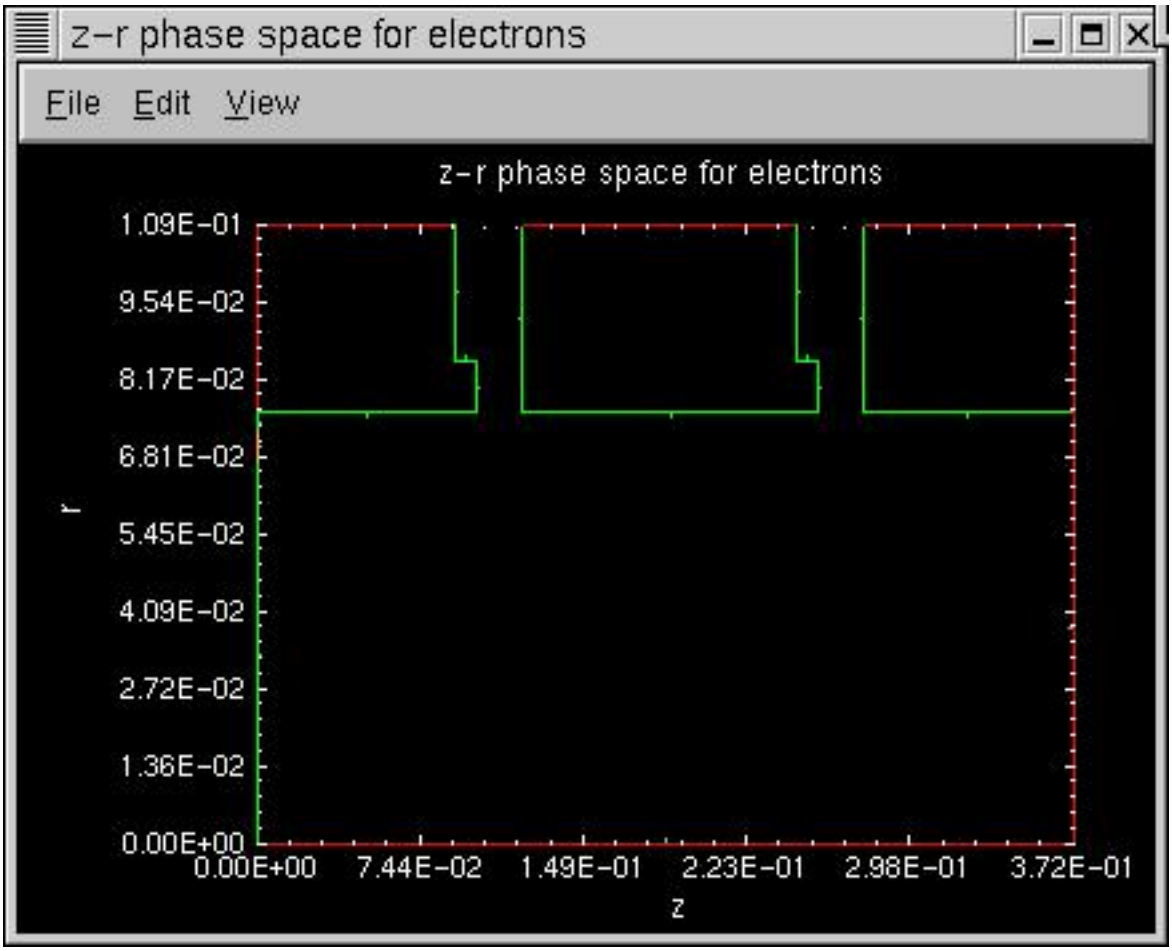

Figure 12: Klystron profile

7.1 A Simple Klystron

This example illustrates the use of OOPIC Pro in modeling the behavior of a simple klystron device.

In the example given here, a two-cavity klystron is modeled in cylindrical coordinates on a 74 by 36 computational grid. The physical dimensions are 0.372 m along the axis of symmetry with a radius of 0.109 m. This diagram shows one half of an axial slice of the klystron that occupies the full computational grid. The full klystron would be a body achieved by rotating this diagram about r = 0. The input RF signal has an amplitude of 30 and a frequency of 1.21 GHz.

The simulation parameters are defined in the file $OOPICPRO/input/klystron5.inp.

The graphs that follow are selected from the OOPIC Pro diagnostics list via the View→ Diagnostics menu. They represent the state of the Klystron after 1500 time steps (about 7.5 nanoseconds). Figure 13 shows the position of the electron beam (shown in blue) as it emerges from the `BeamEmitter` *component and impinges on the opposite wall of the klystron (traveling from left to right in this plot). Keep in mind that this diagram only shows one half of an axial slice so that to visualize the klystron in three dimensions this figure would have to be rotated about the axis of symmetry. This means that the electron beam itself is actually shaped like a hollow, thin-walled cylinder within the klystron. The bunching of electrons along the beam path is visibly evident in this plot near the input and output ports.*

The influence of the modulating wave entering the klystron through the input cavity can be seen in the electron bunching (nonuniform density of blue points representing electron macroparticles) shown in Figure 13 and also in the phase plot shown in Figure 14. This figure plots the scaled axial velocity of electrons against their axial position. In

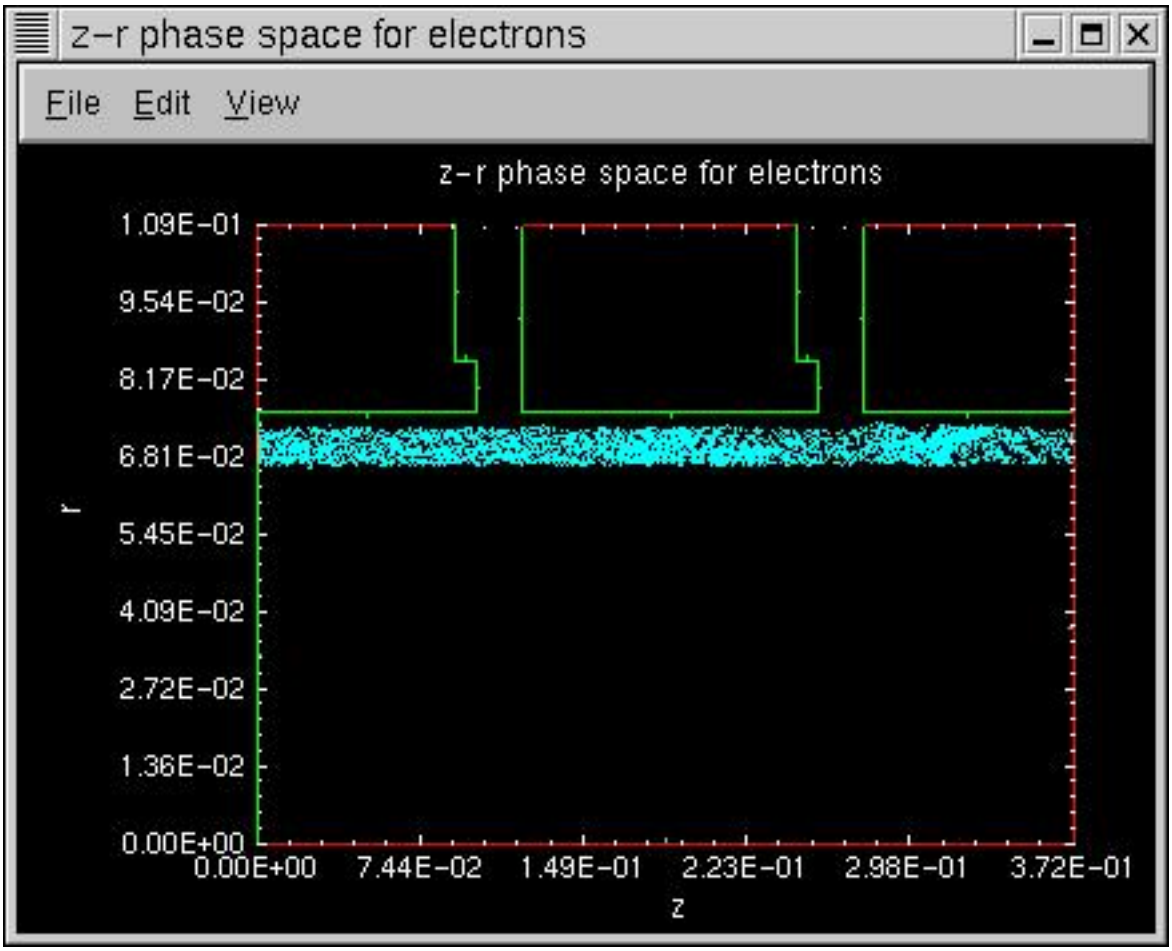

Figure 13: Cross-section of electron beam traversing the klystron

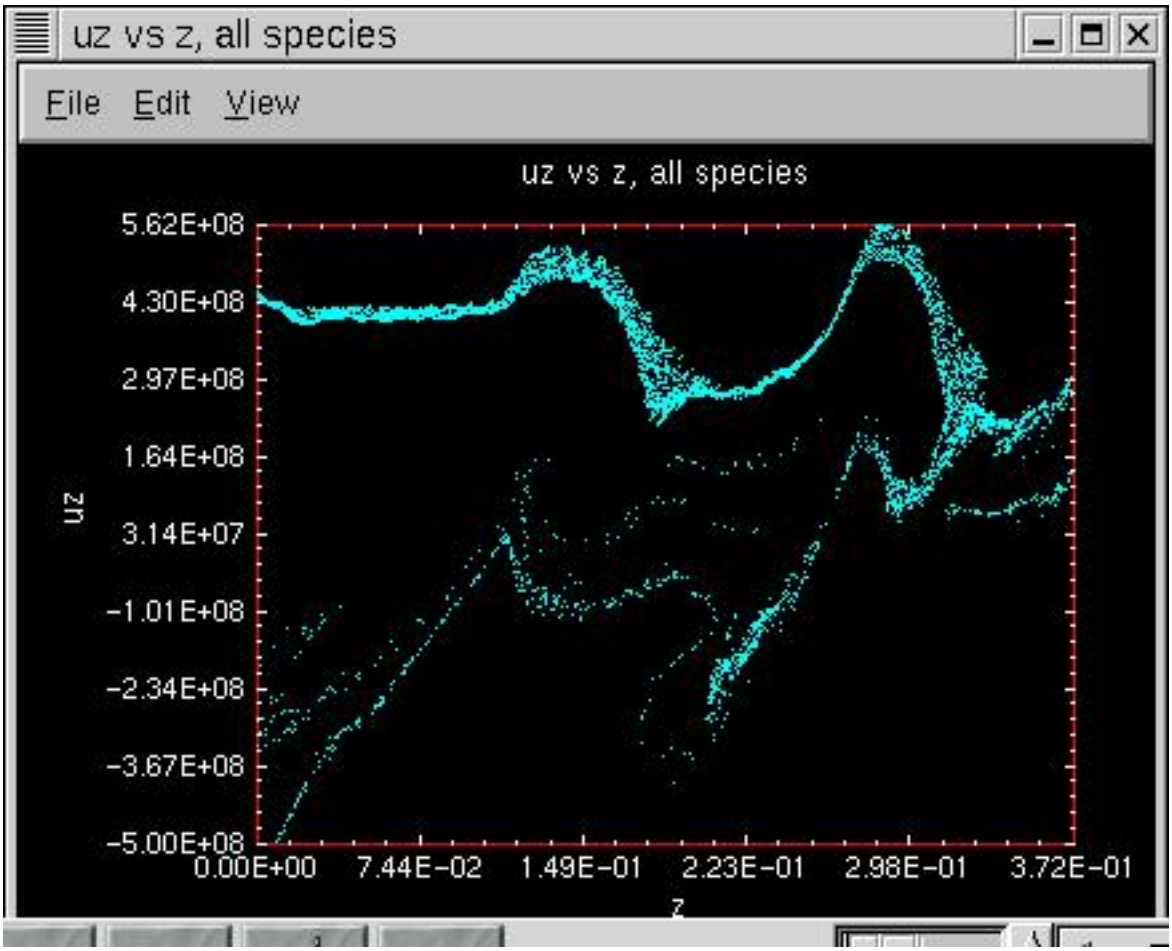

Figure 14: Scaled electron velocity as a function of axial displacement

this example the input drift velocity of the electrons is 2.480e8 m/s. the scaled velocity is $u = gv$ *where*

$$g = (1 - v^2/c^2) - 1/2 = 1.780$$

By this point in time in the simulation the beam of electrons has been able to traverse both the input and the output port and impinge on the opposite klystron wall as well. Passing the input cavity subjects the electrons to the time-dependent forcing function which has been specified in the input file. The forcing function alternately slows and accelerates electrons that pass near the input cavity. This leads to the velocity profile seen in Figure 14 in which some electrons have a negative velocity along the axis and it can be seen that a general circulation (or apparent bunching) of electrons is occurring within the klystron.

Also of interest are history traces of the Poynting vector at the Input and Output cavity openings. Collection of this data is controlled by turning on the EFFlag parameter in the input file. At the Input Cavity/Klystron interface the variation in the Poynting vector is initially due to only the modulated input RF signal. Eventually electron circulation contributes as well.

At the Output Cavity, the Poynting vector magnitude (Figure 16) has a more regular periodic variation with an increasing magnitude. This illustrates the use of a klystron device as an RF amplifier.

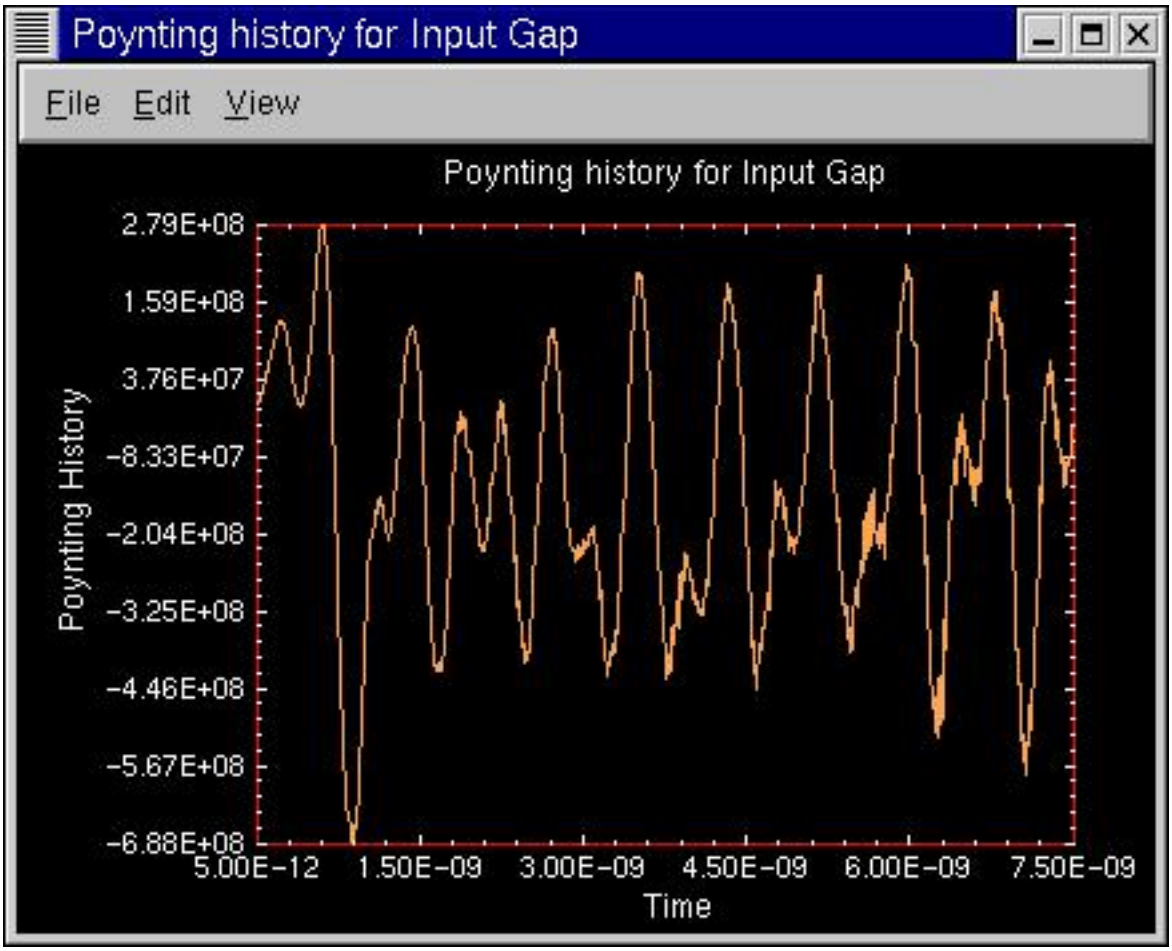

Figure 15: Poynting vector at the Input Cavity interface

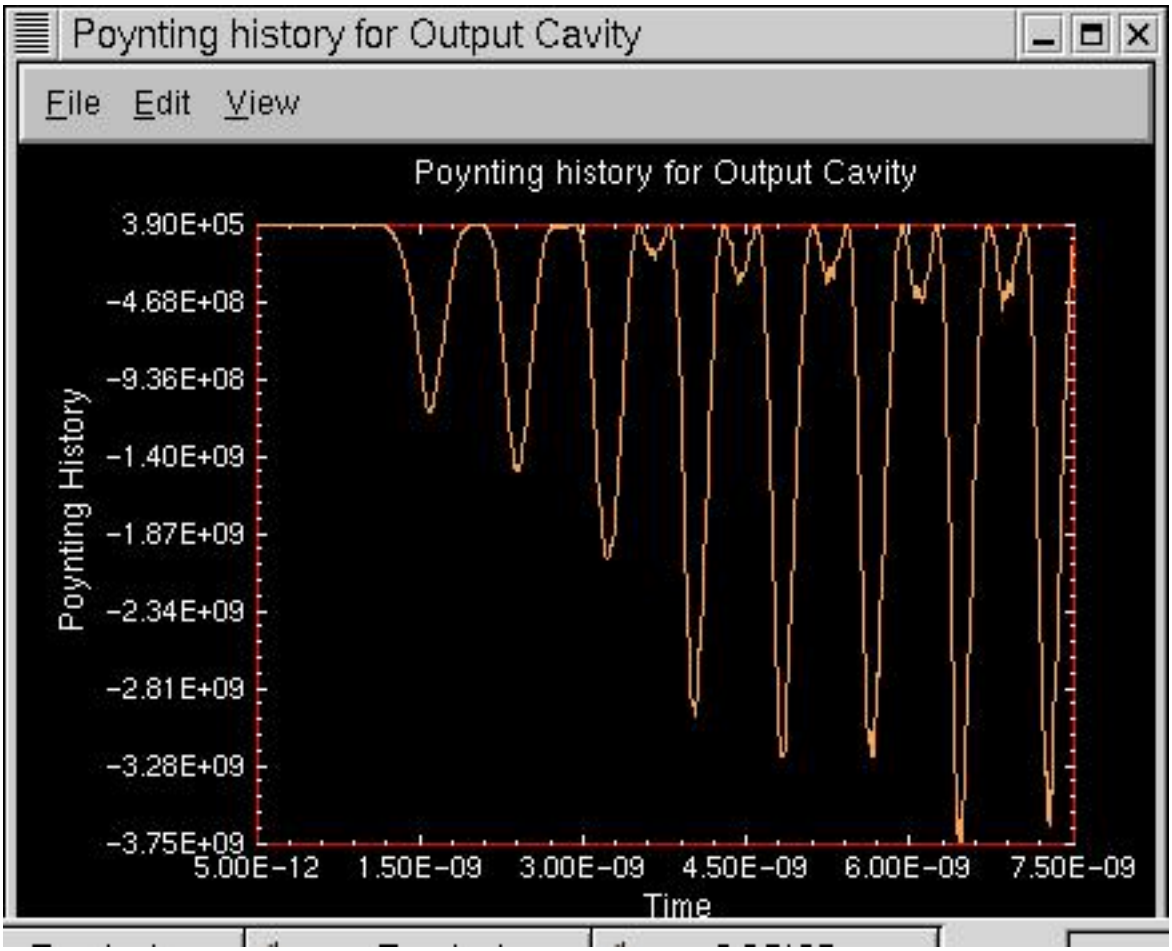

Figure 16: Poynting vector at the Output Cavity interface

7.2 Field-induced Tunneling Ionization by a Laser Pulse

This example ($OOPICPRO/input/loasisHe.inp) illustrates the use of OOPIC Pro in modeling tunneling ionization due to propagation of a laser pulse into and through a neutral gas; Helium in this case. This example has been used to study phenomena such as the single and double ionization of Helium by laser impact and frequency up-shifting of the leading edge of the laser pulse. For more detail refer to "Simulations of Laser Propagation and Ionization in l'OASIS Experiments" and "Particle-in-cell simulations of plasma accelerators and electron-neutral collisions."

The l'OASIS input file defines a 2D simulation conducted in Cartesian coordinates. The x coordinate defines the longitudinal direction along which the laser pulse will propagate. The polarization of the pulse is in the z direction, the redundant coordinate. The pulse enters a "simulation box" from its left face and propagates to the right. An OOPIC `PortGauss` *element is defined at this boundary to launch the laser pulse. The input file further specifies that, in the transverse direction, the pulse will have a Gaussian shape. The two sides of the simulation box adjacent to the entry face are defined to be* `ExitPorts` *and the opposite face is a* `Conductor`*. Due to symmetries, the simulation box can be represented by a single slice along the z plane that is overlaid with a computational two-dimensional grid 256 by 512 in size.*

With the laser wavelength fixed, the sampling theorem requires that the laser wavelength span at least two grid points. Increasing this ratio improves the spatial resolution of the simulation but needs to be weighed against the computational resources available. A minimum time step can be computed from requirements to satisfy the Courant condition for adequate resolution of phenomena in the time domain once this set of geometric parameters has been established.

For this example, the laser wavelength l = 0.8 mm. With a grid size N[x] = 512, N[y] = 256 and 16 grids per wavelength for adequate spatial resolution, the time step is $1.65x10^{-16}$ *sec. See the input file for details of this calculation. Most of the geometric parameters have been equated to symbols that are then used to calculate derived parameters such as the time step required to satisfy the Courant condition.*

In order to be able to model a realistic experimental domain OOPIC Pro provides a moving window capability that supports the adjustment of the numerical grid to center on physical phenomena of interest as the simulation progresses in time. In this example the geometry of the simulation is initially set up to reflect the simulation box that will be required to fully contain a single laser pulse for a finite time before it approaches the boundaries of a helium cloud (with sharp edges). As the laser penetrates the helium, the geometric boundaries of the simulation, as maintained by OOPIC Pro, shift along the direction of propagation in order to more accurately capture the physical phenomena occurring in and around the laser pulse. OOPIC Pro continues to move the simulation box along the direction of pulse propagation until the pulse has completely exited the helium cloud and the interaction is complete.

For this example, the density of helium changes from 0 to $2x10^{25}m^3$ *over a distance of four grid cells, eight cells from the initial end of the simulation box. The moving window algorithm is activated when the pulse is within 15 cells of the end of the simulation box. The helium cloud extends a distance of two Rayleigh wavelengths along the propagation axis.*

Figures 17 and 18 show the laser pulse in the first phase of the simulation. This reflects the time just prior to activation of the moving window algorithm when the laser pulse is about to impinge upon the helium cloud.

OOPIC Pro's moving window allows the definition of the simulation box to actually extend beyond the immediate computational grid by following the laser pulse and ignoring the rest of the system for the computations that need to be performed to capture the current laser/gas interaction.. This keeps computational load manageable and robust. The computational grid remains at the same manageable size but will cover sequential sections of the simulation box as it follows the propagation of the laser pulse. In the graphic output it will be seen that coordinates along the x direction do not change with time. This is because the coordinates shown relate to the moving window and not the full simulation box.

At time t = 0 in the simulation the pulse is launched into a vacuum and propagates along the x direction into the simulation box. It eventually encounters the neutral He gas. The gas density profile is a linear ramp increasing quickly over a small number of grid cells along x from zero (vacuum) to the maximum density specified in the input file. The gas density ramps downward in a similar fashion after a distance along x. This allows the pulse to travel for some distance while totally immersed in the neutral gas. The pulse then exits the simulation through a vacuum at the far end

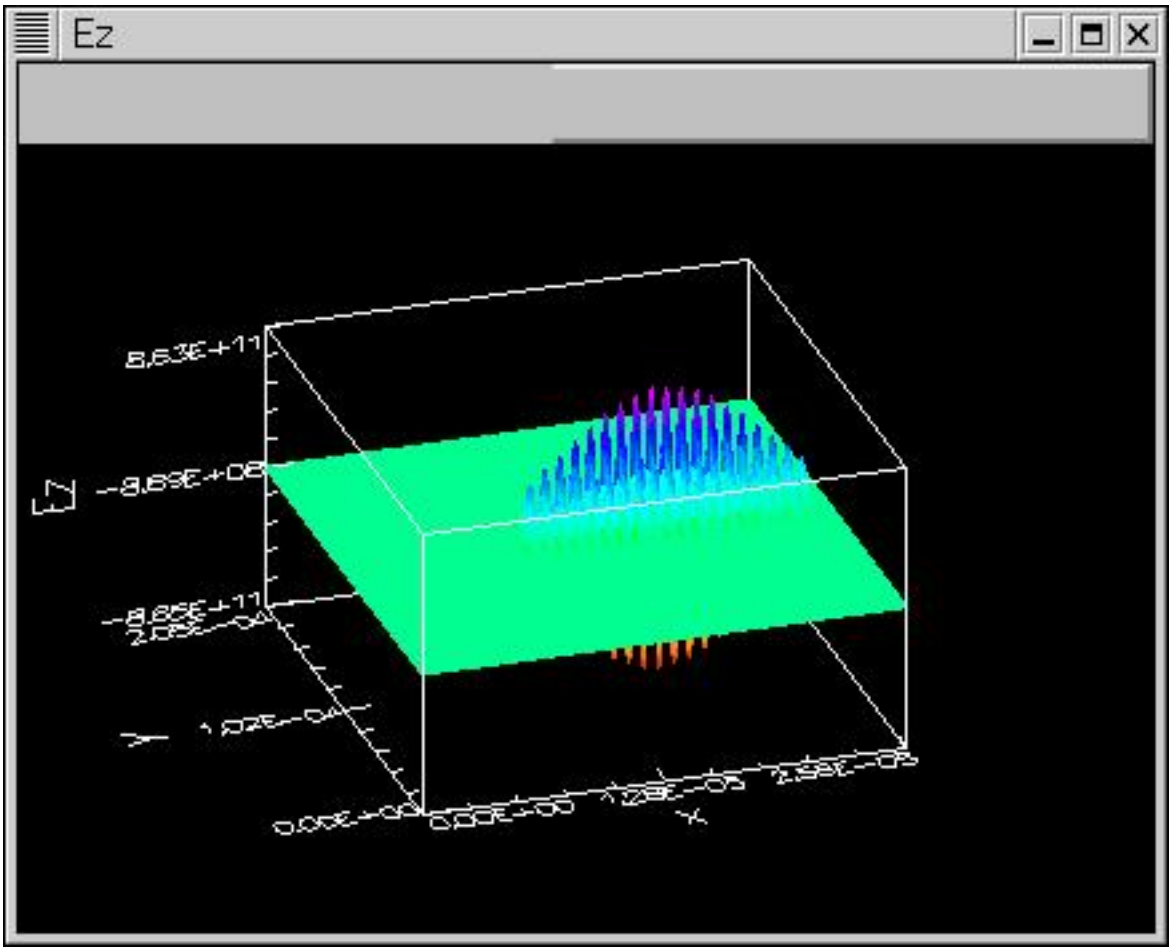

Figure 17: Laser pulse profile in vacuum

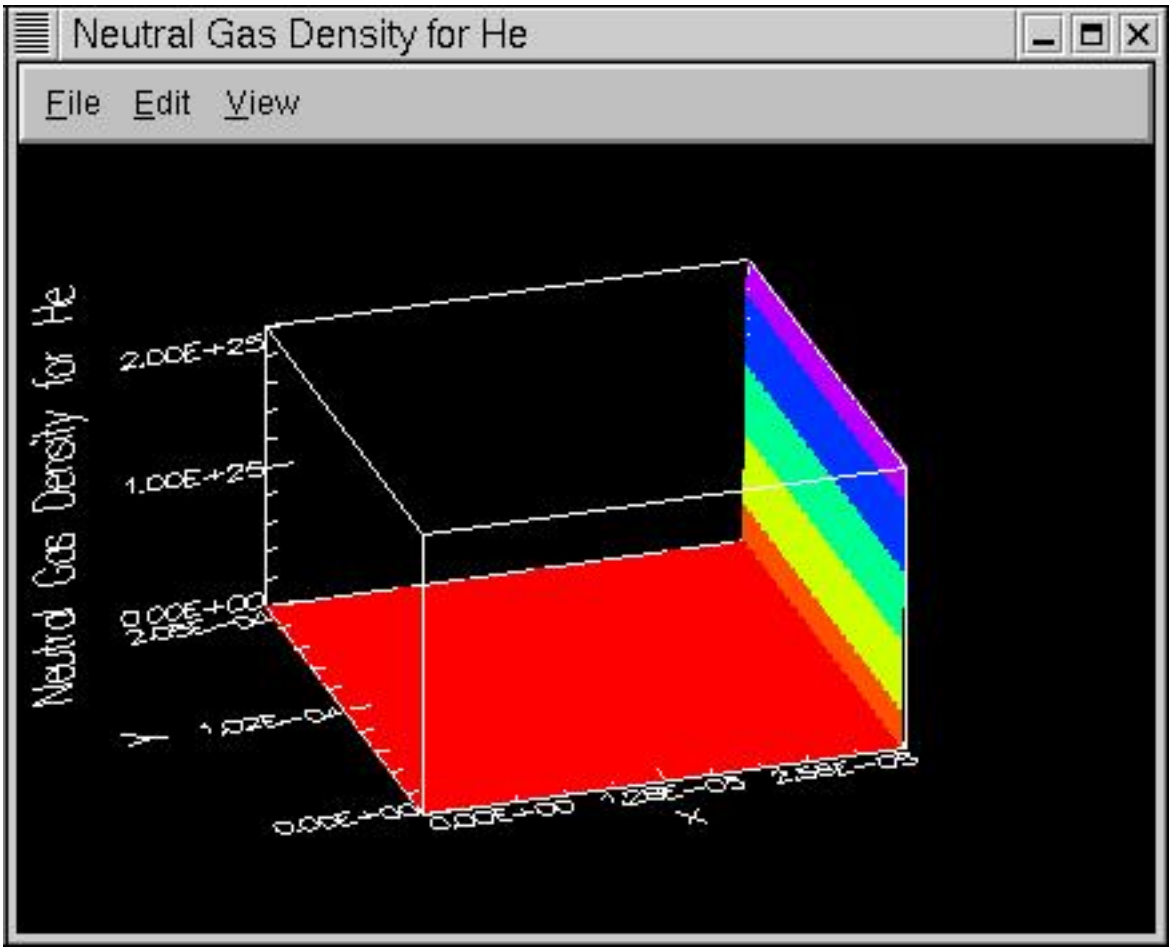

Figure 18: Profile of Helium density in the area of vacuum and just inside the region of helium cloud

of the simulation box.

While the pulse propagates through the neutral gas, it can ionize the Helium in its path, creating He^+, and can also ionize the He^+, creating He^{++}. Ionization in this situation occurs through the quantum phenomena of tunneling ionization (see "Particle-in-cell simulations of plasma accelerators and electron-neutral collisions." for a more complete explanation of tunneling ionization). The computation of tunneling ionization effects is included in the simulation by turning on the tunnelingIonizationFlag parameter in the `MCC` element of the `Region` block of the input file.

The electron, gas and ion densities shown by this next set of snapshots of the simulation extend, as might be expected, to the point where the laser pulse is about to exit the region of He gas. The pulse has left a trail of fully ionized helium in the form of He^+ and He^{++}.

Once the laser pulse exits the helium cloud densities for all species will drop to zero.

Once the laser pulse has exited the helium cloud completely it propagates through a vacuum once again.

7.3 Plasma Wakefield Accelerator

This example ($OOPICPRO/input/beamplasma.inp) illustrates the use of OOPIC Pro in modeling a beam-driven plasma wakefield accelerator (PWFA). It is similar to the case discussed in Section V of Particle-in-cell simulations of plasma accelerators and electron-neutral collisions published in Phys. Rev. Special Topics — Accel. & Beams, Issue

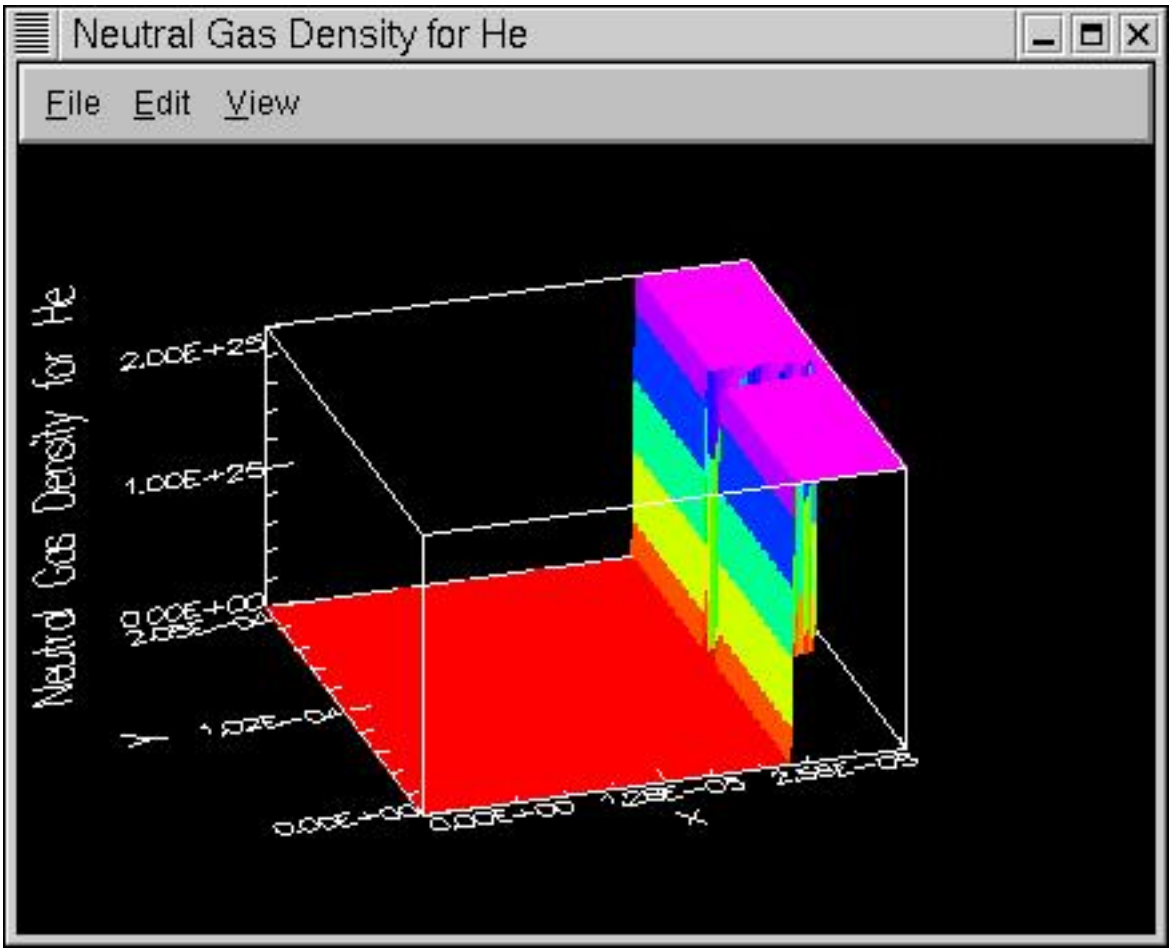

Figure 19: As the laser pulse enters the helium cloud it ionizes all He within its path. The volume within the profile of the laser pulse is essentially void of any neutral He. Tunneling ionization occurs on the edge of the laser pulse where there is a steep gradient in He density.

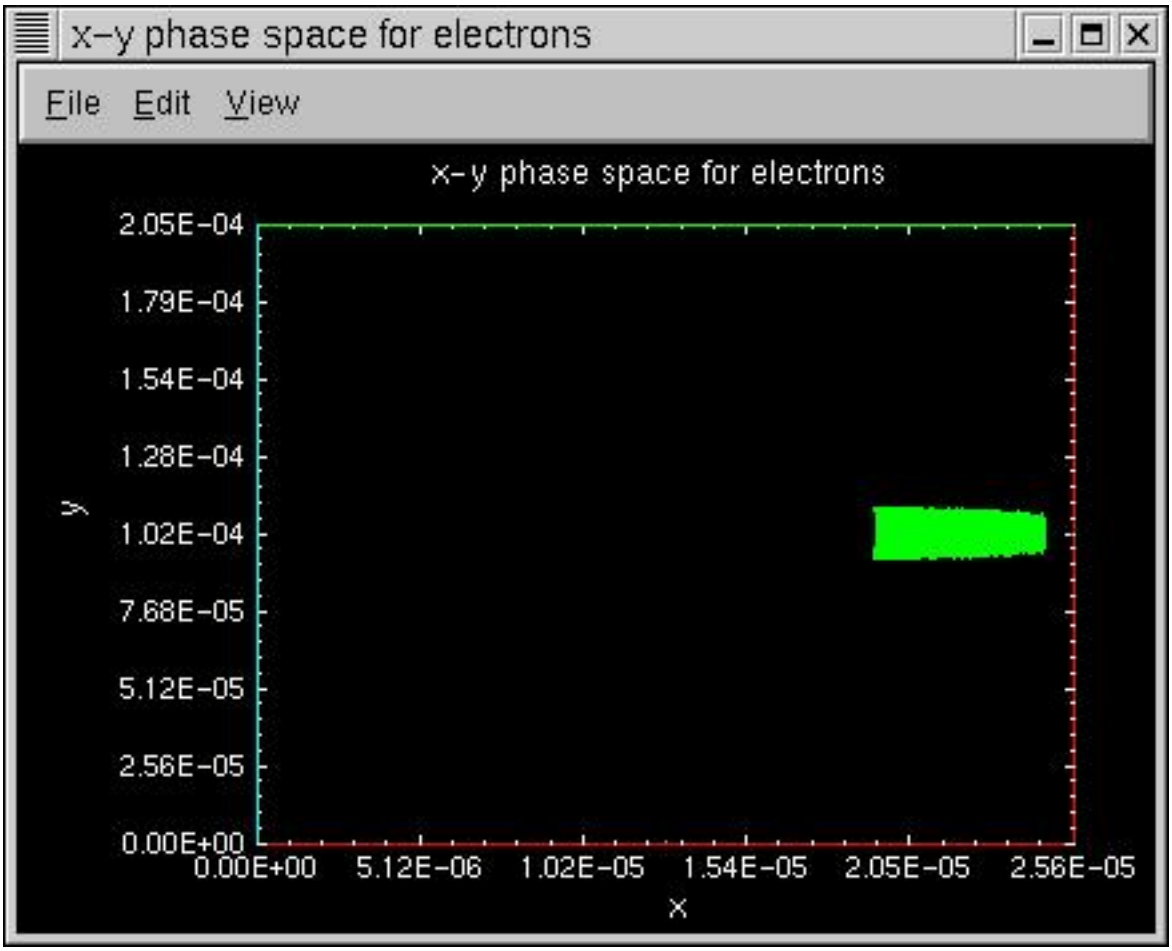

Figure 20: Electrons freed by He and He^+ ionization conform to the profile of the laser pulse.

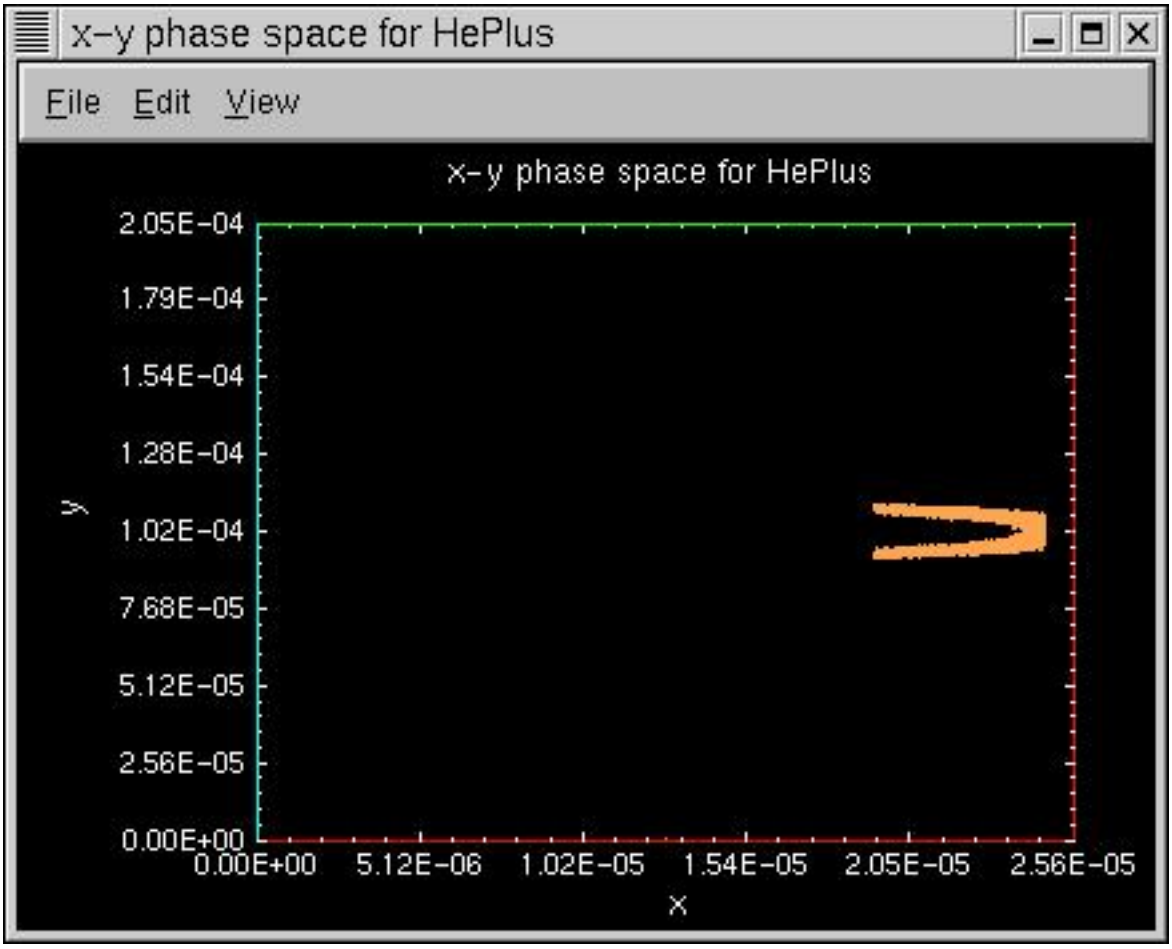

Figure 21: The core of the laser pulse is void of any He^+

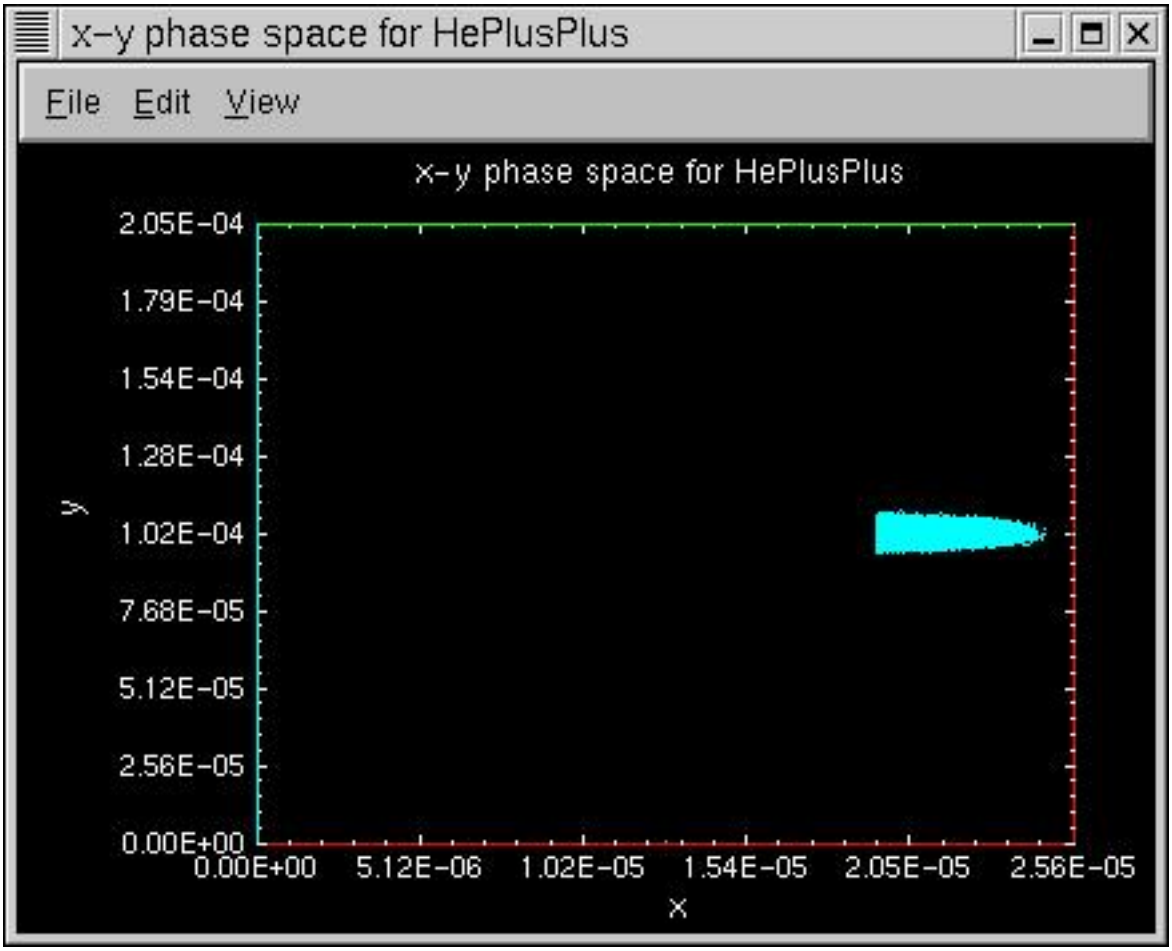

Figure 22: He^{+} at the core of the laser pulse is further ionized into He^{++} since the pulse has sufficient intensity there but not at its edges.

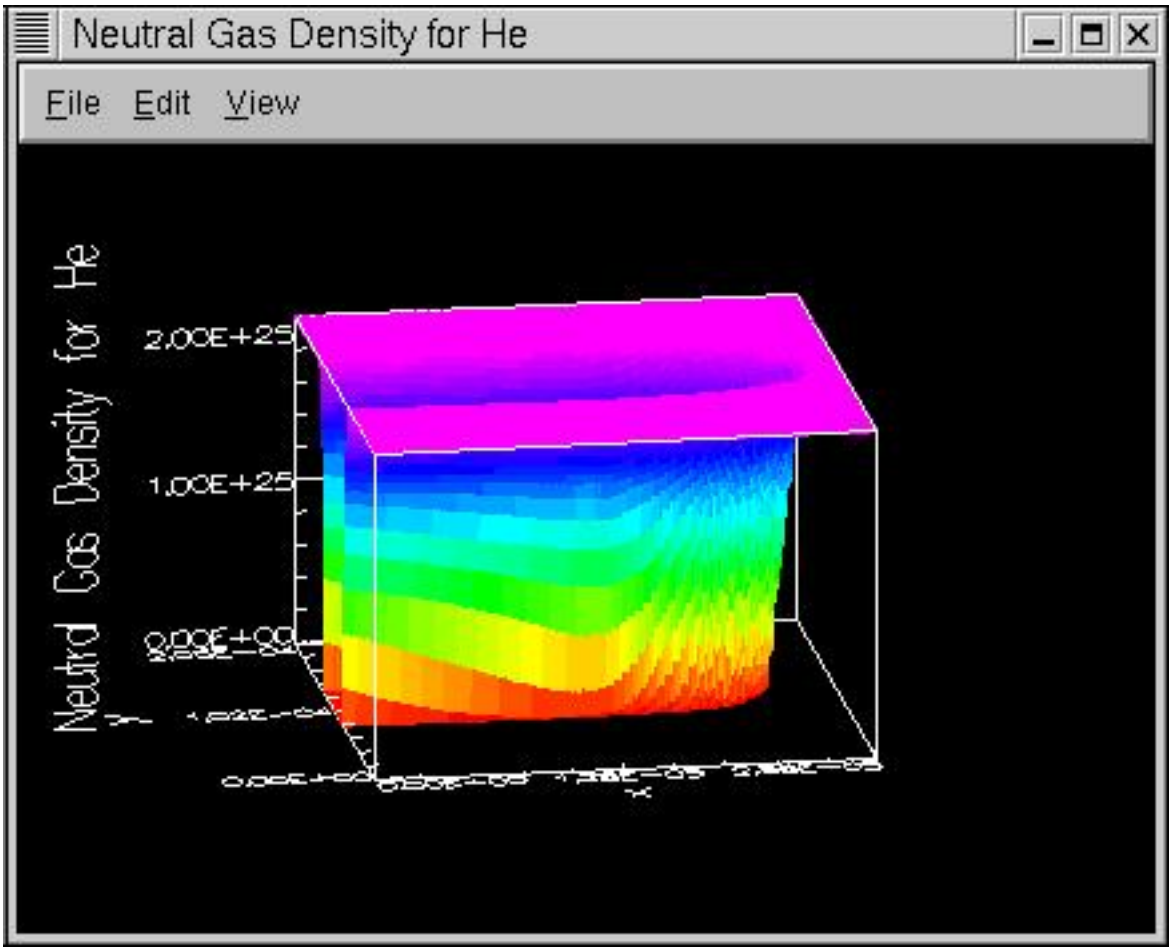

Figure 23: Profile of Helium density in the midst of helium cloud.

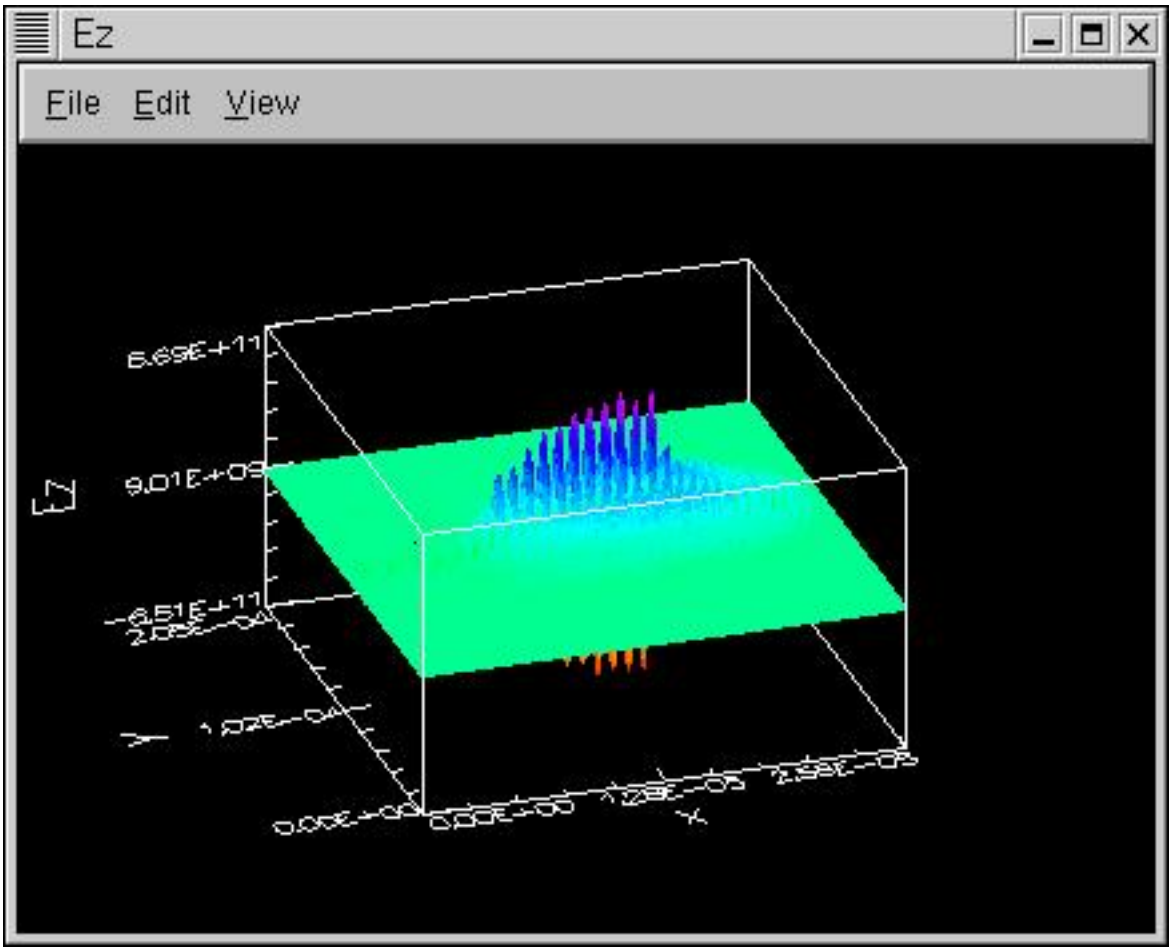

Figure 24: Laser pulse profile in Helium cloud

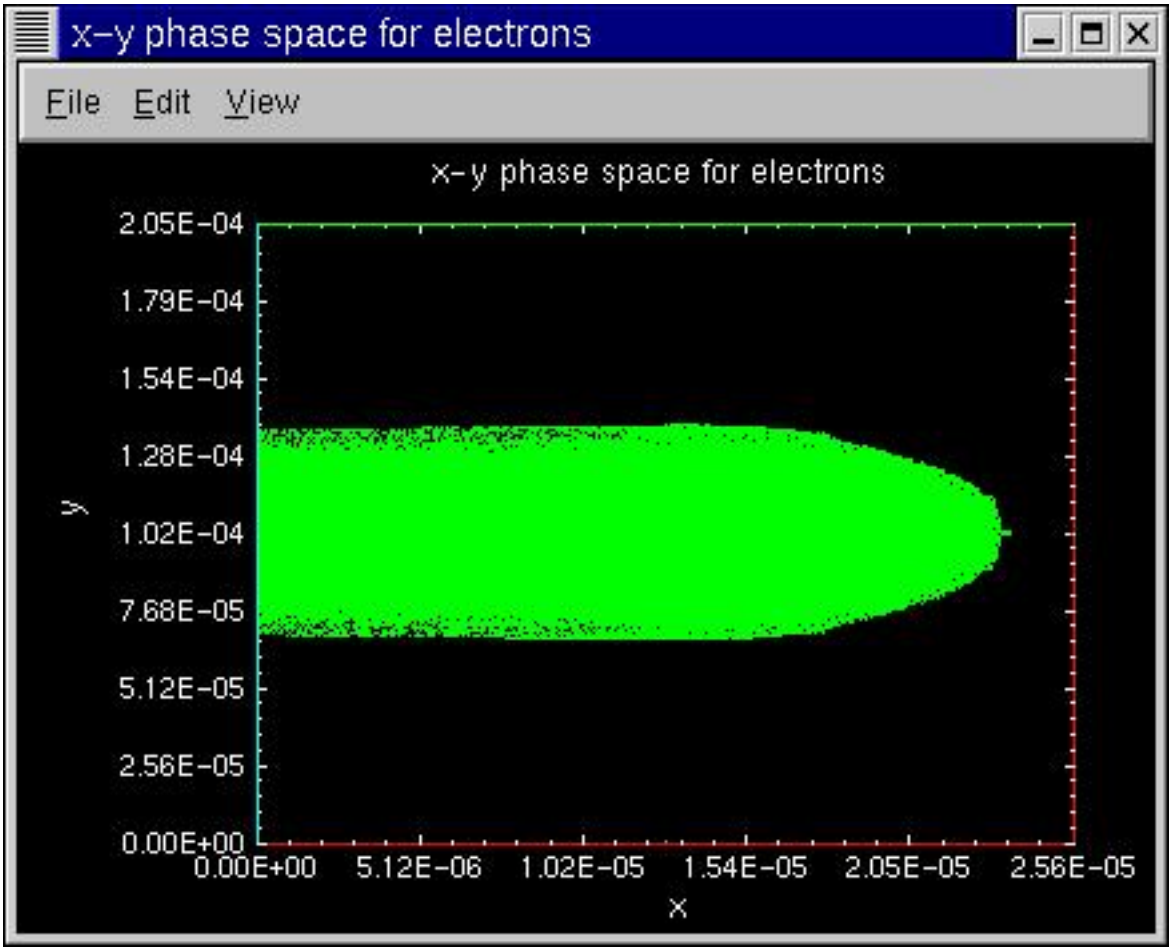

Figure 25: Electrons freed by He and He^+ ionization. The width of the region where free electrons occur is not only longer due to penetration of the laser pulse but also wider than it was at points near its initial entry.

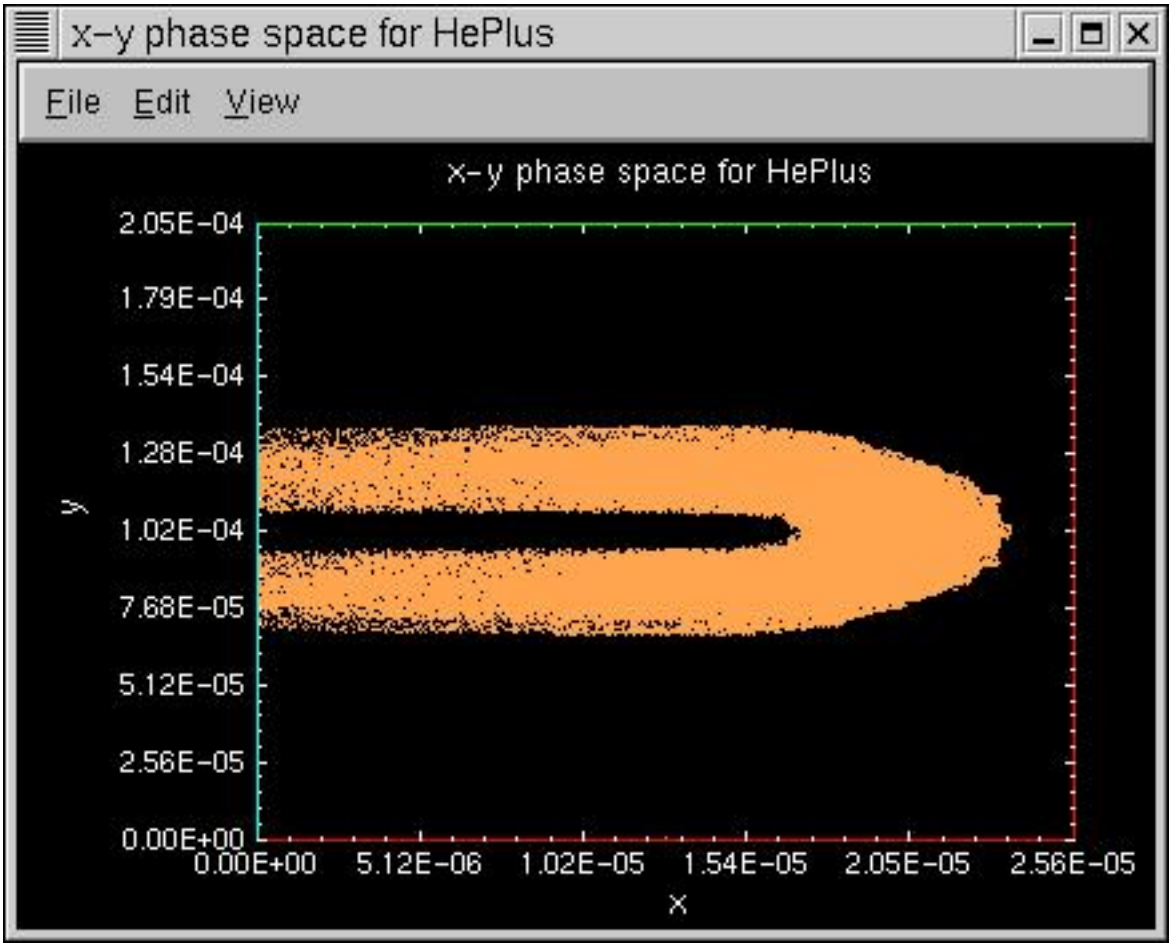

Figure 26: He^{+} ion density. As in the earlier snapshot, He^{+} is ionized to form He^{++} in the interior region of the laser pulse.

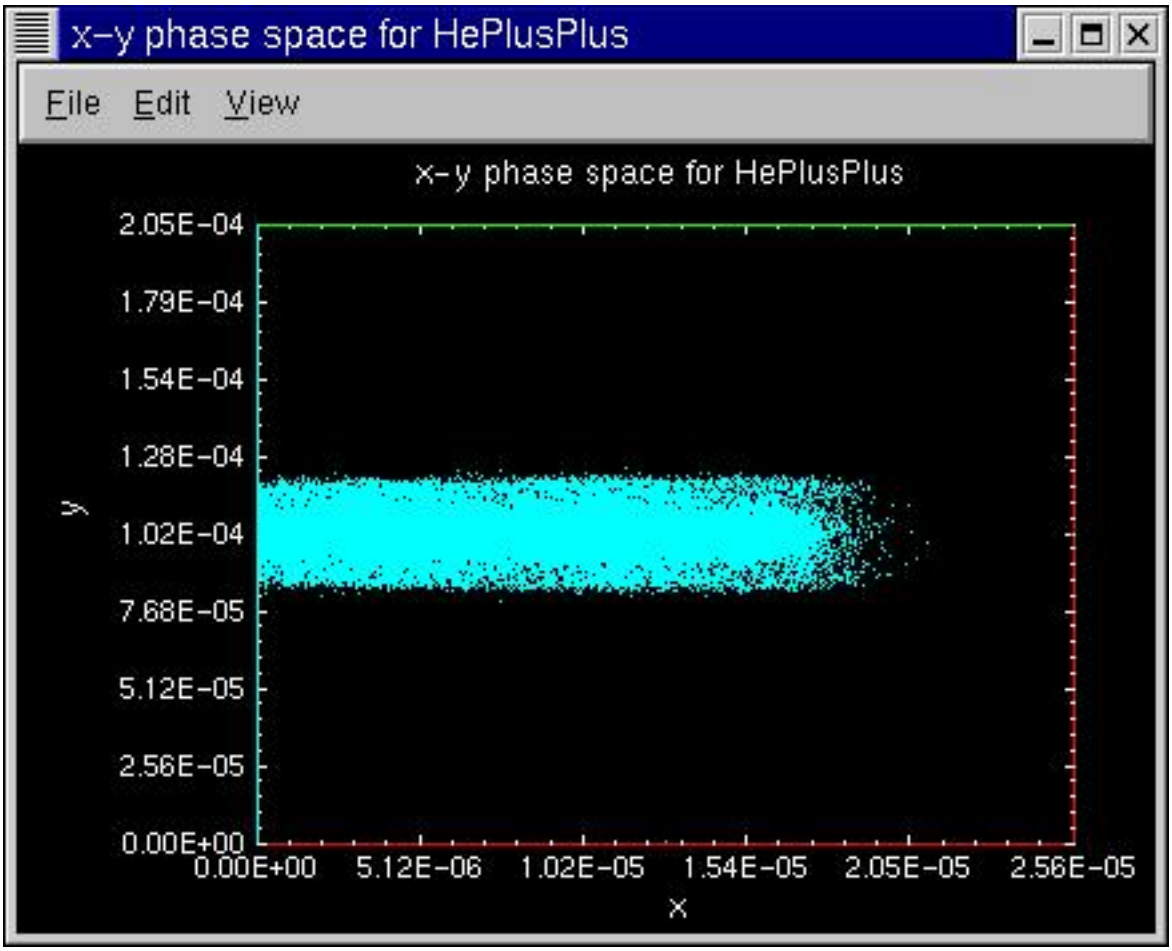

Figure 27: He^{++} ion density

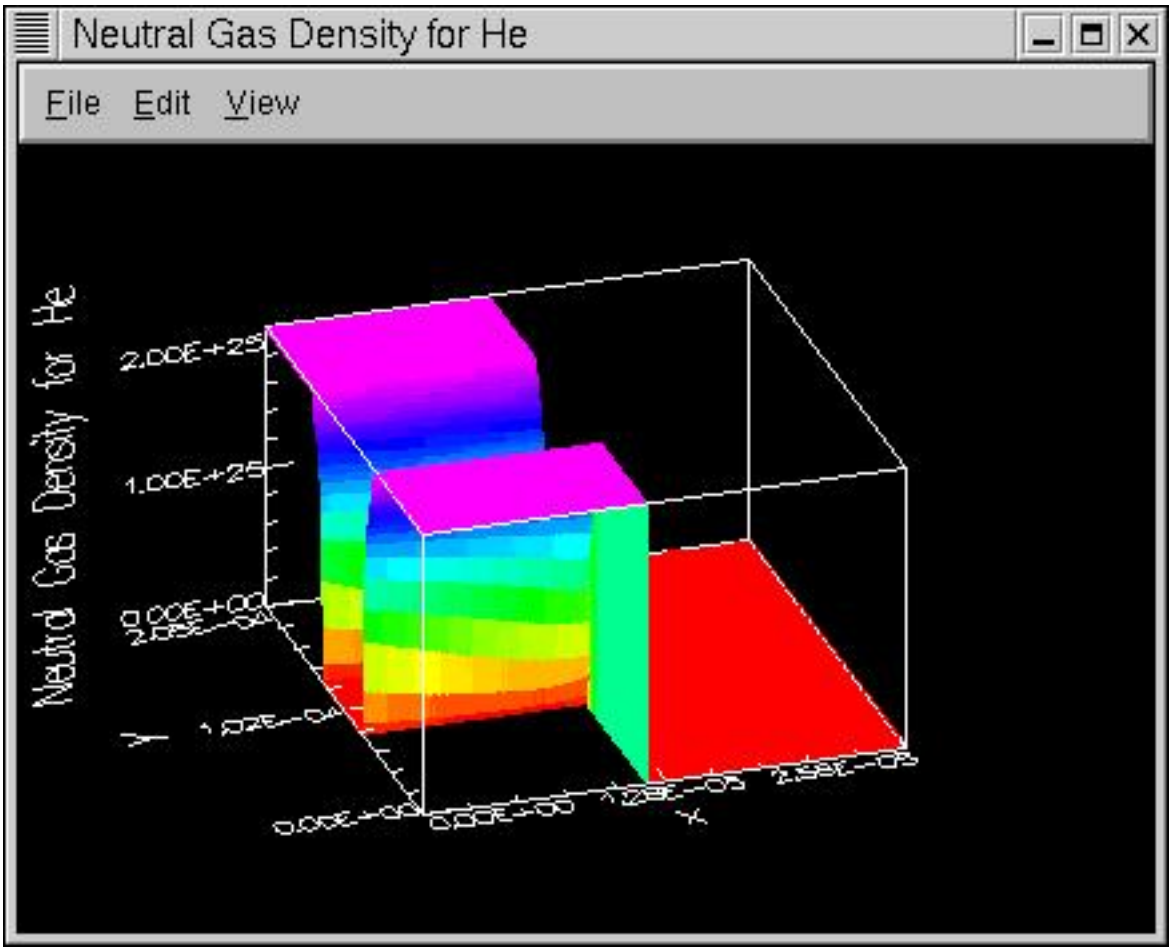

Figure 28: He density profile as laser pulse exits helium cloud.

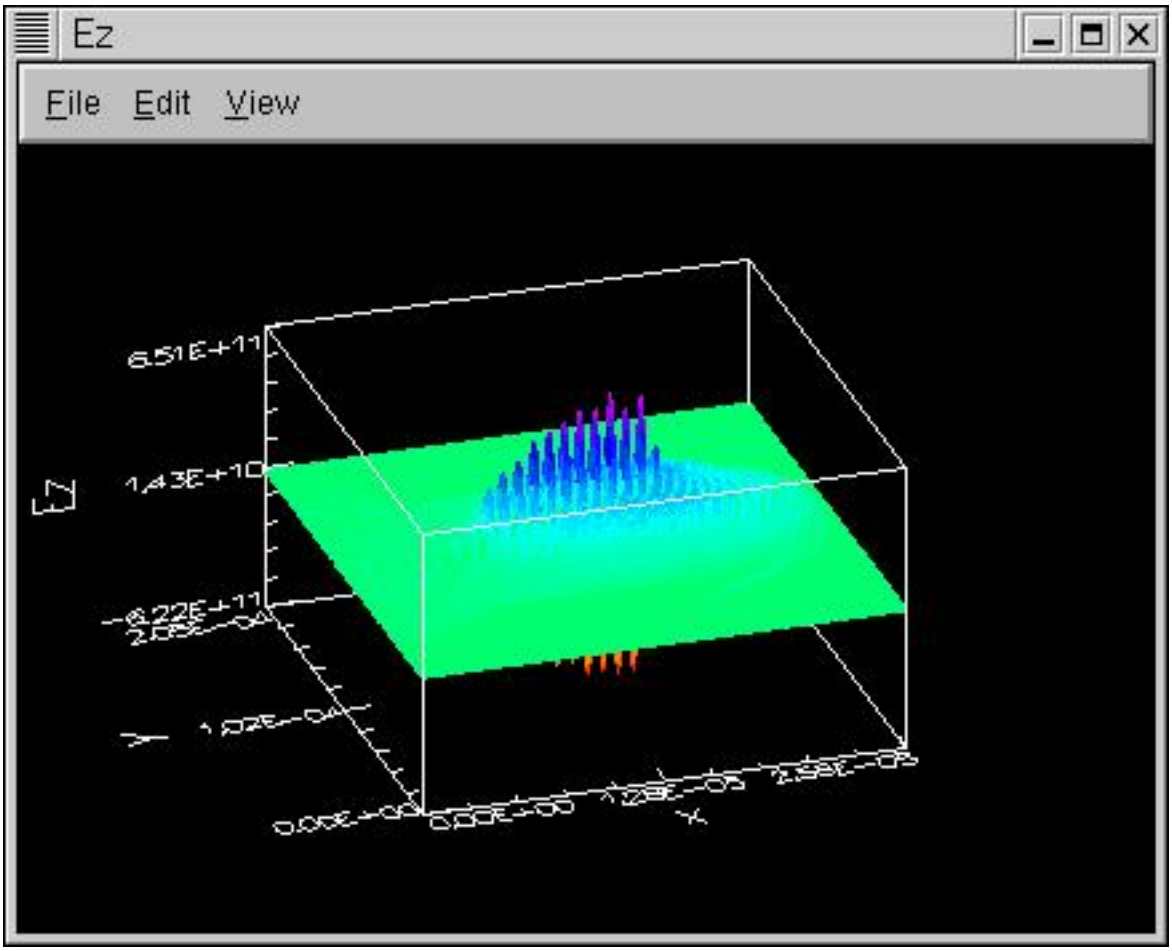

Figure 29: Electric field of laser pulse exiting helium cloud.

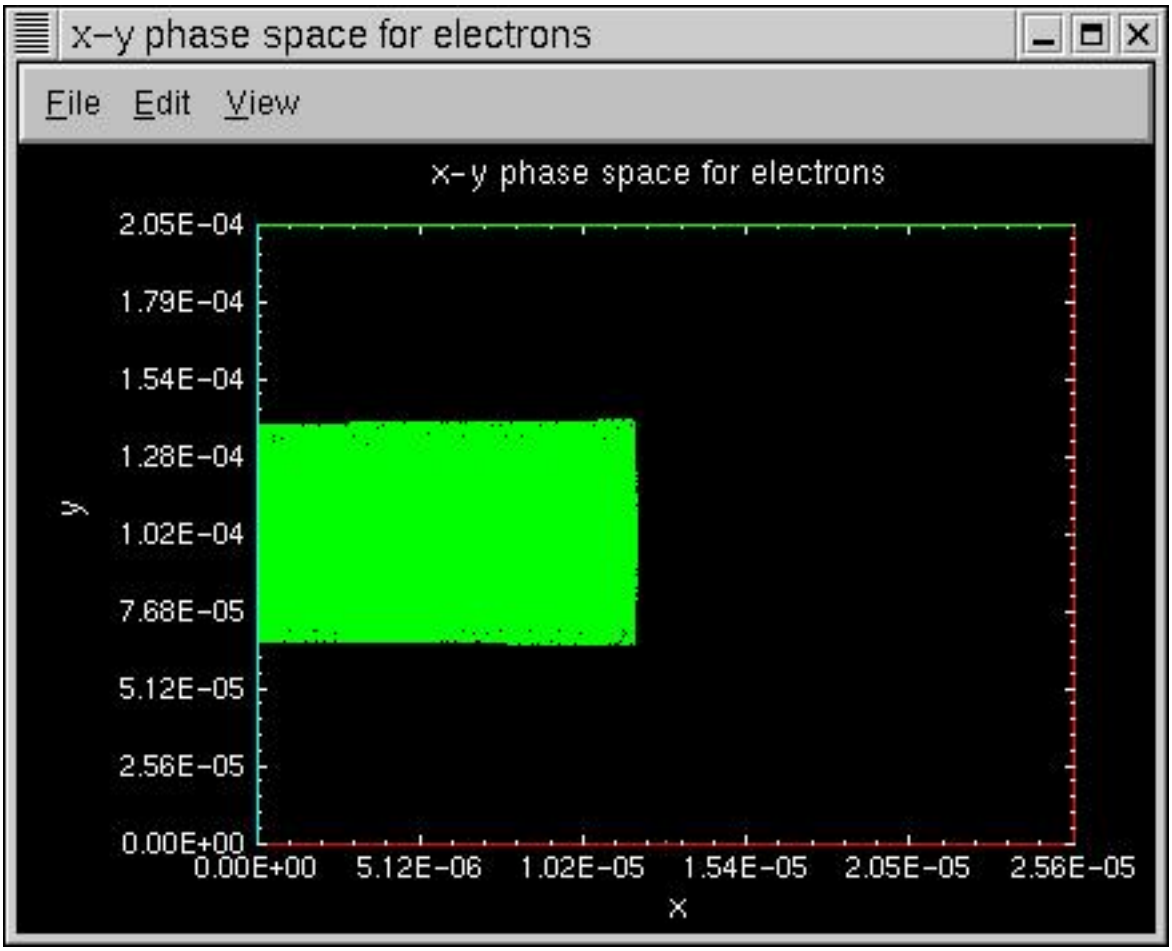

Figure 30: Electron density within helium cloud as laser pulse exits.

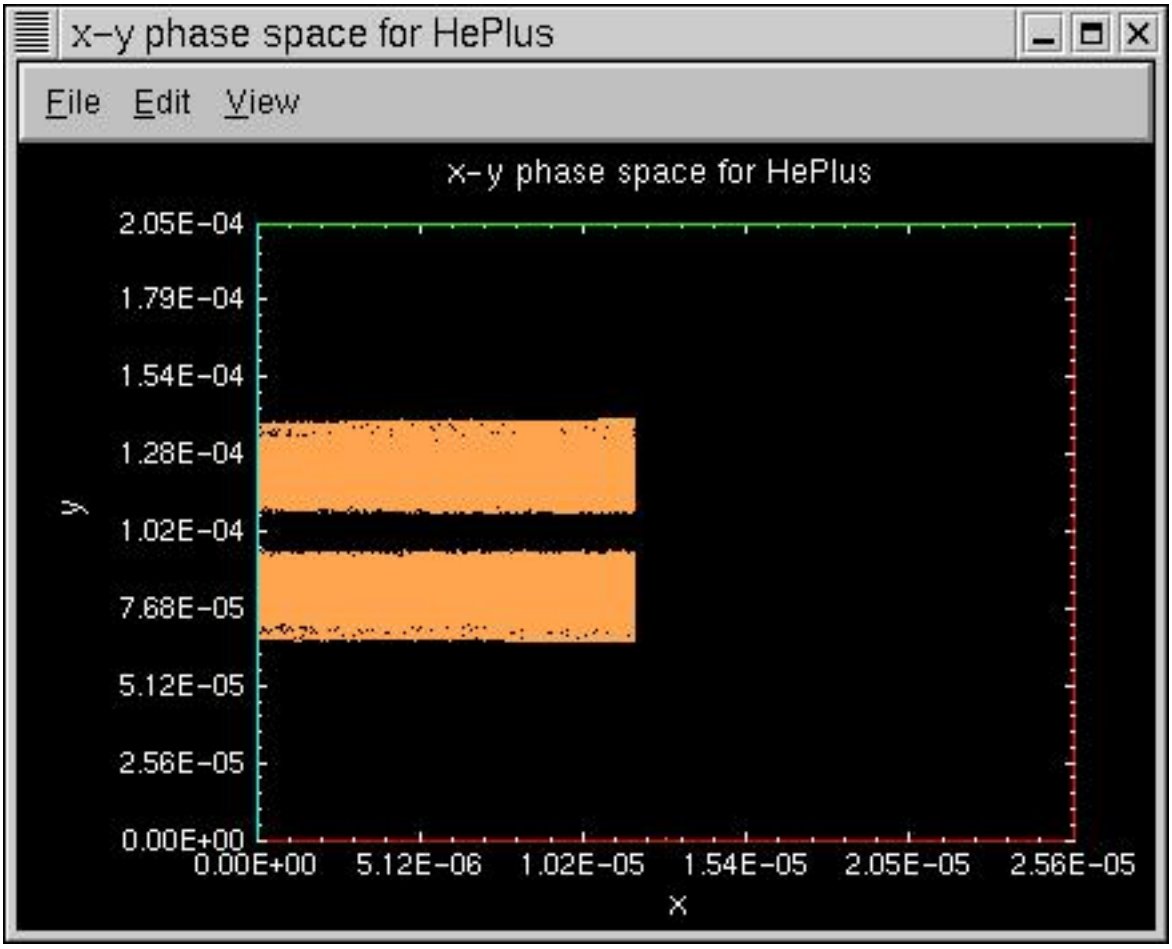

Figure 31: He^+ ion density within helium cloud as laser pulse exits.

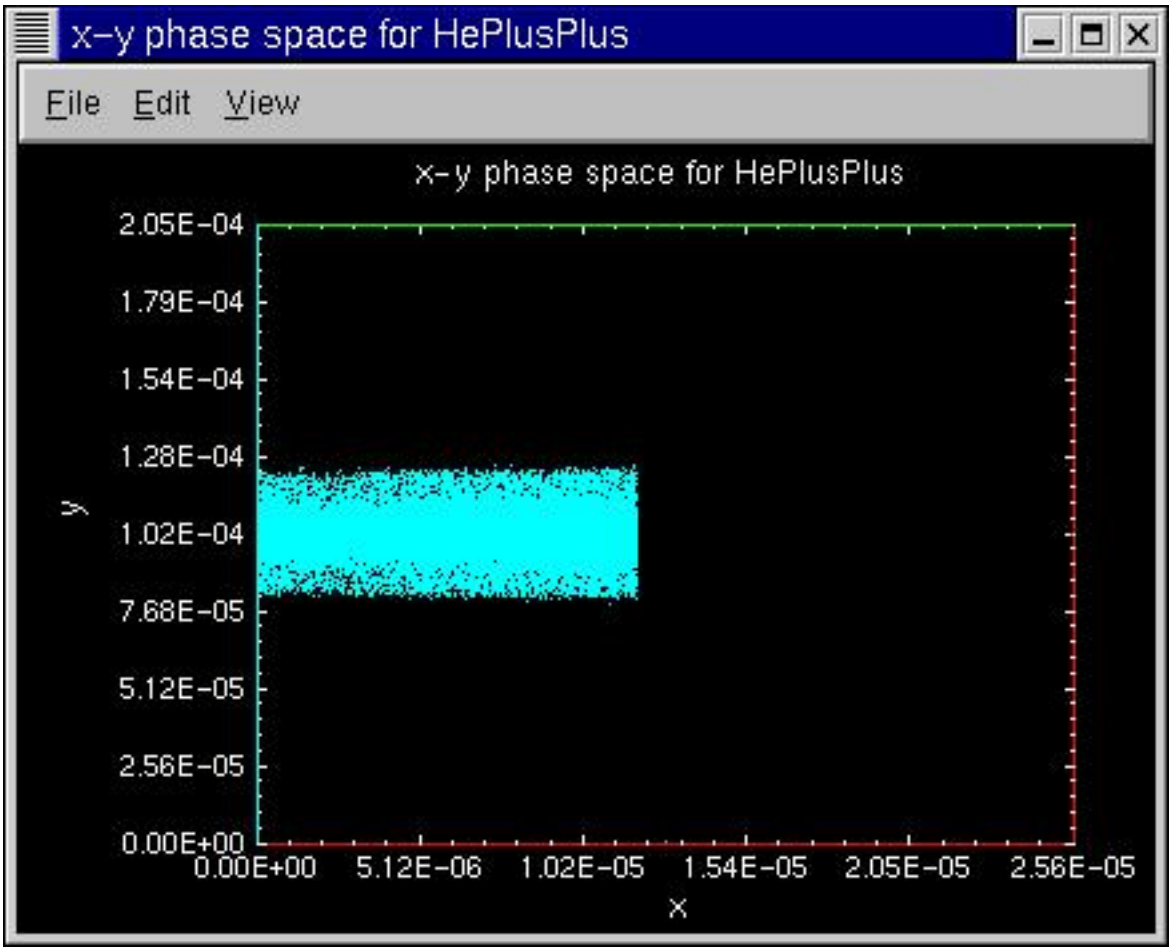

Figure 32: He^{++} ion density within helium cloud as laser pulse exits.

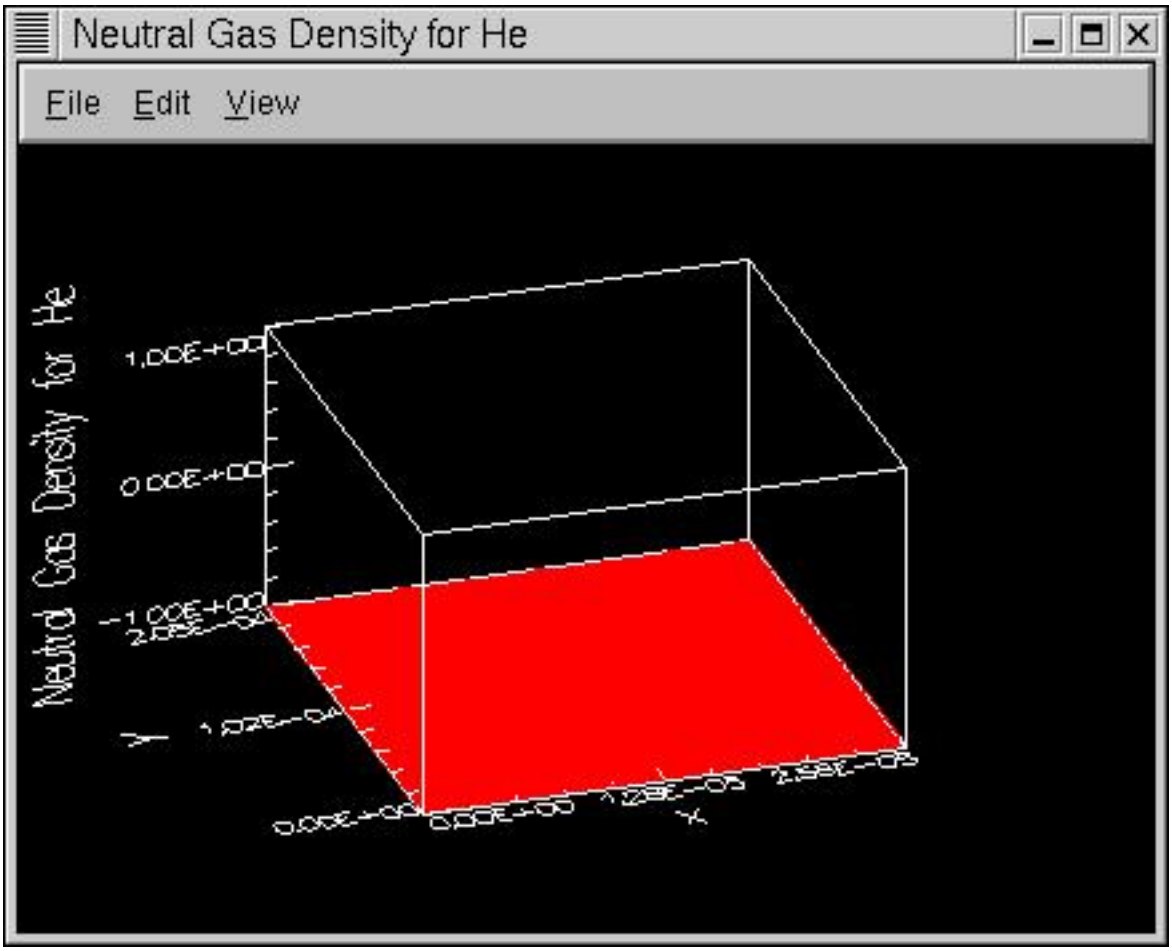

Figure 33: Helium density profile (zero) beyond the helium cloud.

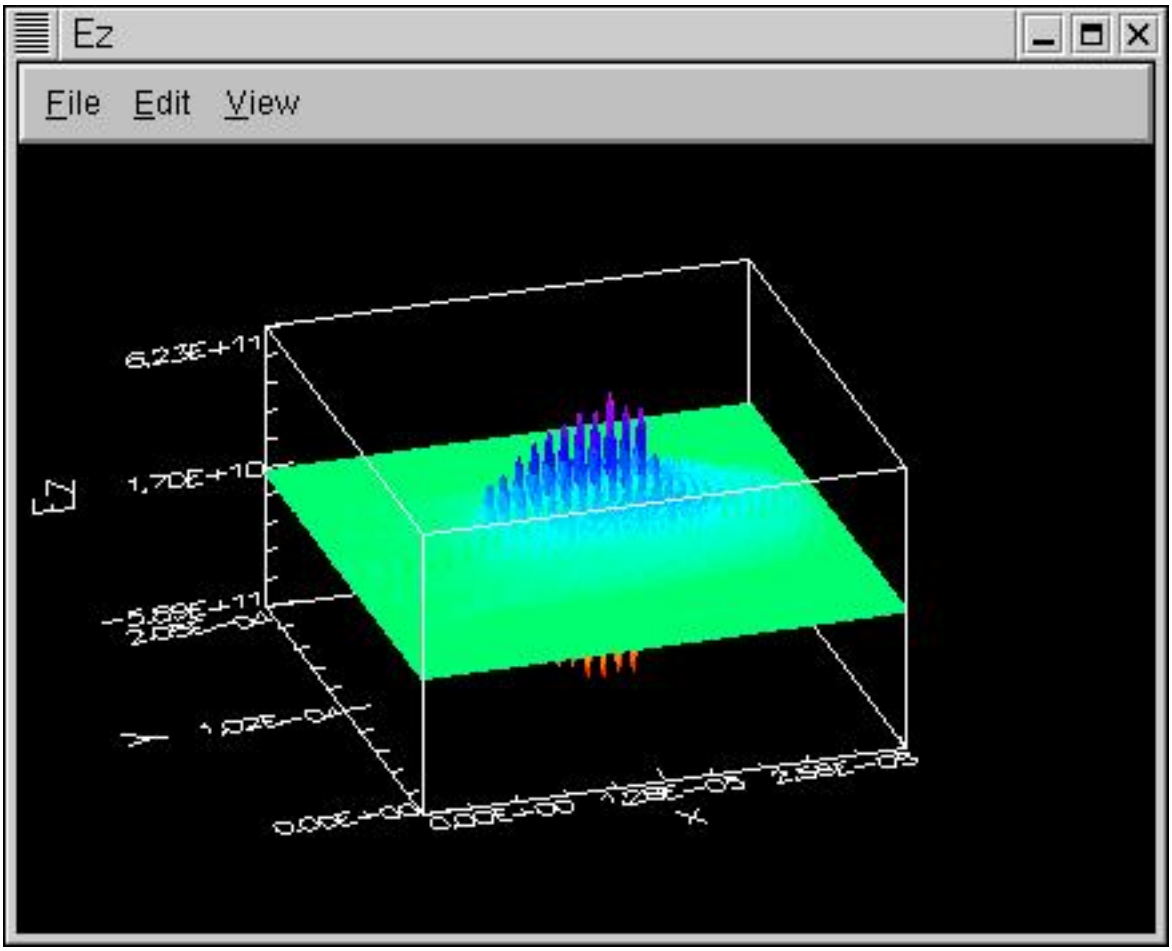

Figure 34: Electric field of laser pulse beyond extent of helium cloud.

10 (October 2001). This report is available as part of the OOPIC Pro documentation set at http://www.txcorp.com/products/oopic.

The input file for this example defines a 2D simulation conducted in cylindrical coordinates. z defines an axis of symmetry along which a short pulse of electrons will propagate. The pulse enters a "simulation box" from its left face and propagates to the right. An OOPIC `BeamEmitter` *element is defined along part of this boundary, centered on the axis of symmetry, to launch the electron pulse. The remaining boundaries of the simulation box are defined to be perfect* `Conductors` *except for the section along the z axis which is established by the CylndricalAxis element. The conducting boundaries are far enough away from the electron beam that they have no effect on the phenomena occurring in its vicinity. The simulation box is represented by a single slice through the axis of symmetry and is overlaid with a computational, two-dimensional grid.*

The example calls for the electron beam to have a maximum radial extent on the grid of eight cells. The ratio of the pulse radius to the model radius is four so the computational grid will have 32 cells in the radial direction. Similarly, the ratio of grid extent in the z direction to that of the radial direction is six so there will be 192 cells along z.

The spatial extent of the grid is computed from values given for the electron beam characteristics. The rms beam width is given to be $0.75x10^{-4}$ *m and is cut off at 3 times this distance. Using the same ratio of beam radius to the model radius gives a spatial extent in the radial direction of 900 mm. Applying the radial/axial ratio of six again gives an axial extent of 5400 mm.*

Geometric spacing on the mesh is chosen to be uniform in both directions so the cells are actually squares. The time step to be used in the simulation that is within limits set by the Courant condition. Even though the Courant condition will guarantee numerical stability of the integration of the field equations, a value arbitrarily smaller than this is chosen to determine the time step. See the input file for this calculation and the scaling factor chosen.

The `BeamEmitter` *definition further specifies that in the axial direction the electron pulse will have a Gaussian shape and that it will have energy of 30 GeV. The uniform density of the pre-ionized lithium plasma is set at* $i2.1x10^{20}m^3$. *The peak density of the electron beam exceeds that of the background plasma so that the self-fields of the beam will cause "blowout" of the plasma electrons. This will create a wake in the distribution of plasma ions as will be shown.*

In order to be able to model a realistic experimental domain OOPIC Pro provides a moving window capability that supports the adjustment of the numerical grid to center on physical phenomena of interest as the simulation progresses in time. In this example the geometry of the simulation is initially set up to reflect the simulation box that will be required to fully contain the pulsed electron beam and allow it to propagate within the ionized lithium plasma for a short time. As the pulse approaches the boundaries of the simulation, the boundaries will begin to shift along the direction of propagation in order to keep the pulse within the plasma and more accurately capture the physical phenomena occurring in and around the electron pulse. OOPIC Pro continues to move the simulation box along the direction of pulse propagation.

Figure 35 shows the electron beam pulse shortly after it has been fully launched by the `BeamEmitter` *element.*

Figure 36 shows the distribution of plasma electrons ion the presence of the 30 GeV electron pulse as it propagates along the z axis. The wake in the plasma electrons is what drives the EPW. Plasma electrons near the head of the pulse are driven away from the beam but return to the axis near the tail of the beam.

Figure 37 shows the electric field surrounding the electron pulse after the plasma electrons have wrapped around the beam pulse as shown in Figure 36. In the region of the tail of the pulse the density of the plasma electrons has increased by nearly two orders of magnitude from the initial density of $2.1x10^{20}m^3$.

Figure 38 shows the Ez component of the electric field within the simulation box with the electron pulse in the same position as Figures 35 through 14.

As the electron beam continues to propagate very little changes except for the expansion of the wake field in the distribution of the plasma electrons. After being blown out of the region along the propagation axis the plasma electrons collapse back towards the axis after the tail of the electron beam has passed. This is shown in Figure 36 as well as, for a later time, in Figure 39. The moving window algorithm in OOPIC Pro has also been invoked by this time.

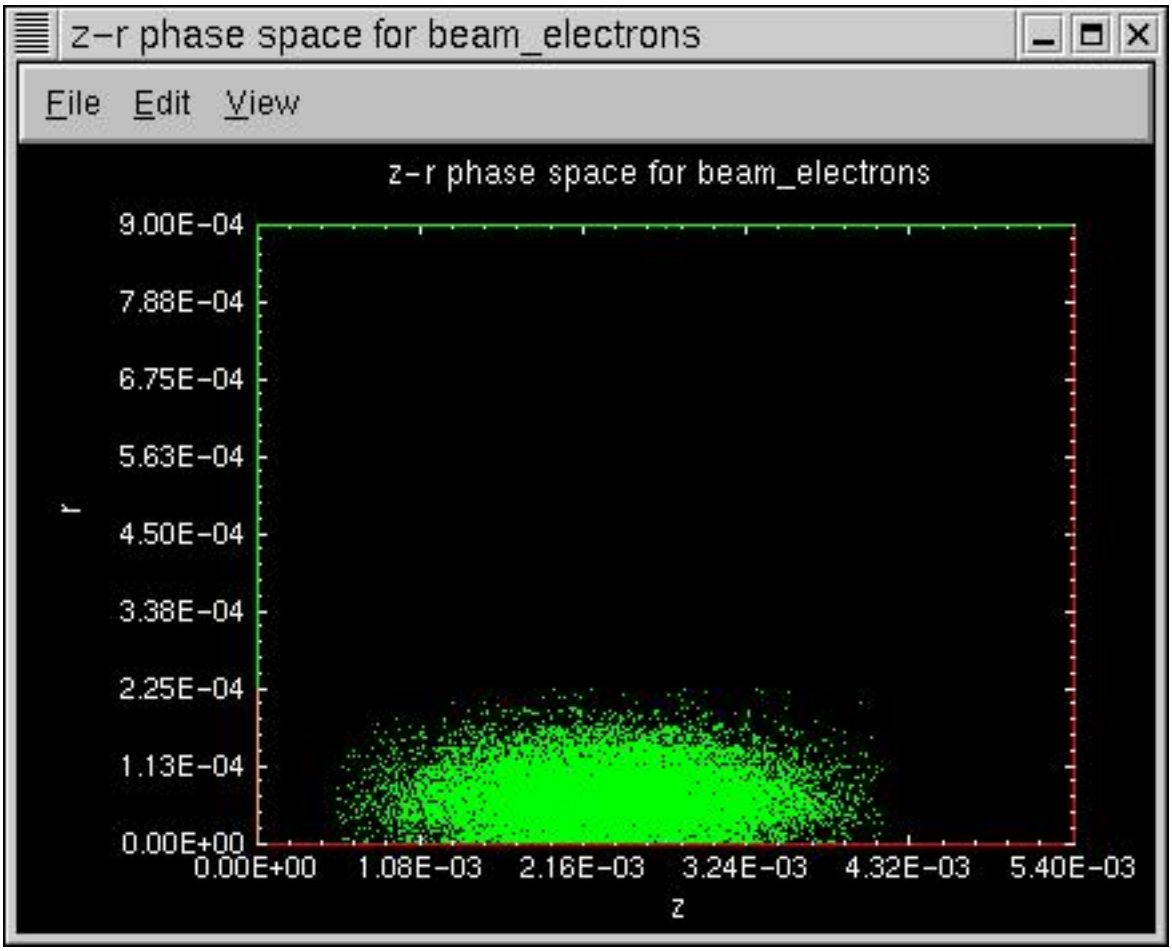

Figure 35: Electron pulse

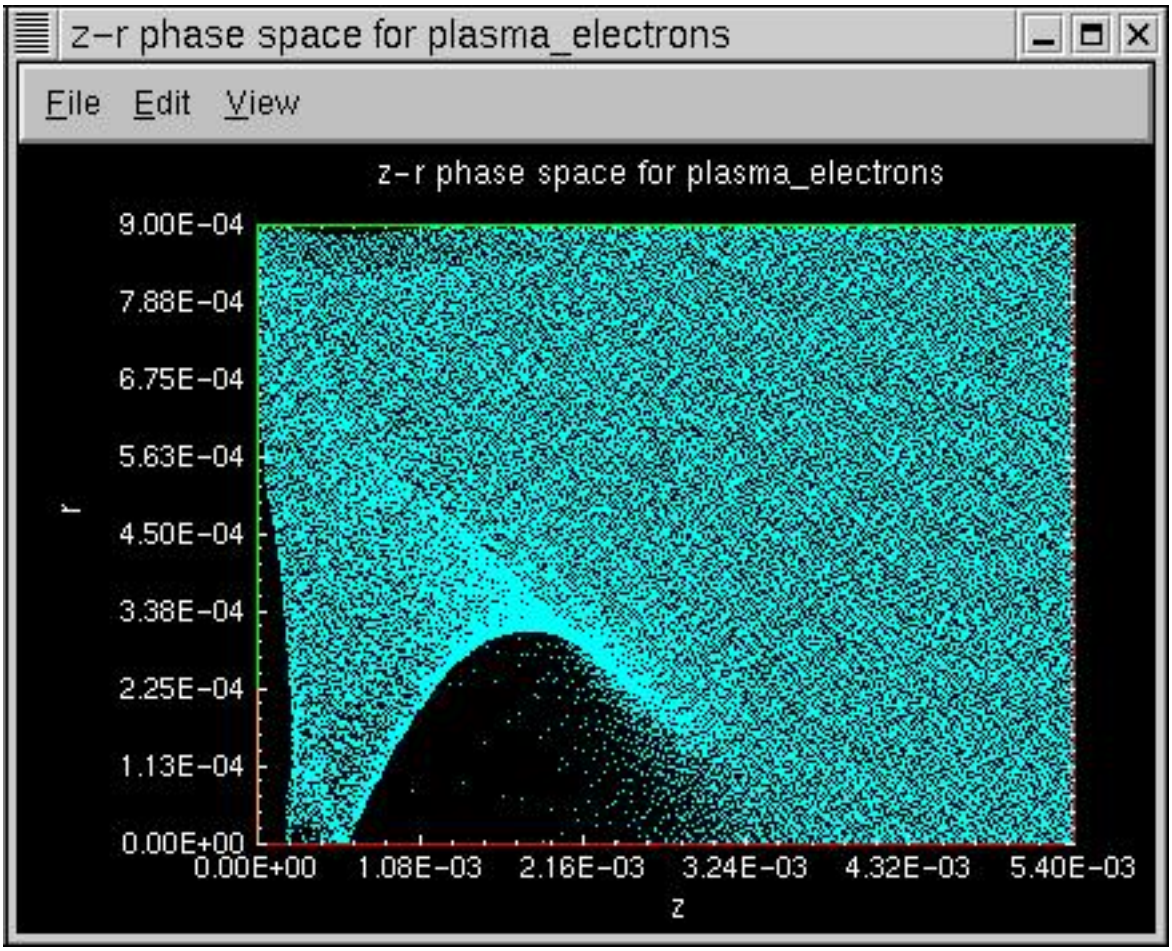

Figure 36: Plasma electron distribution in presence of electron beam showing wake left by electron pulse, $t = 1.38x10^{-11}$.

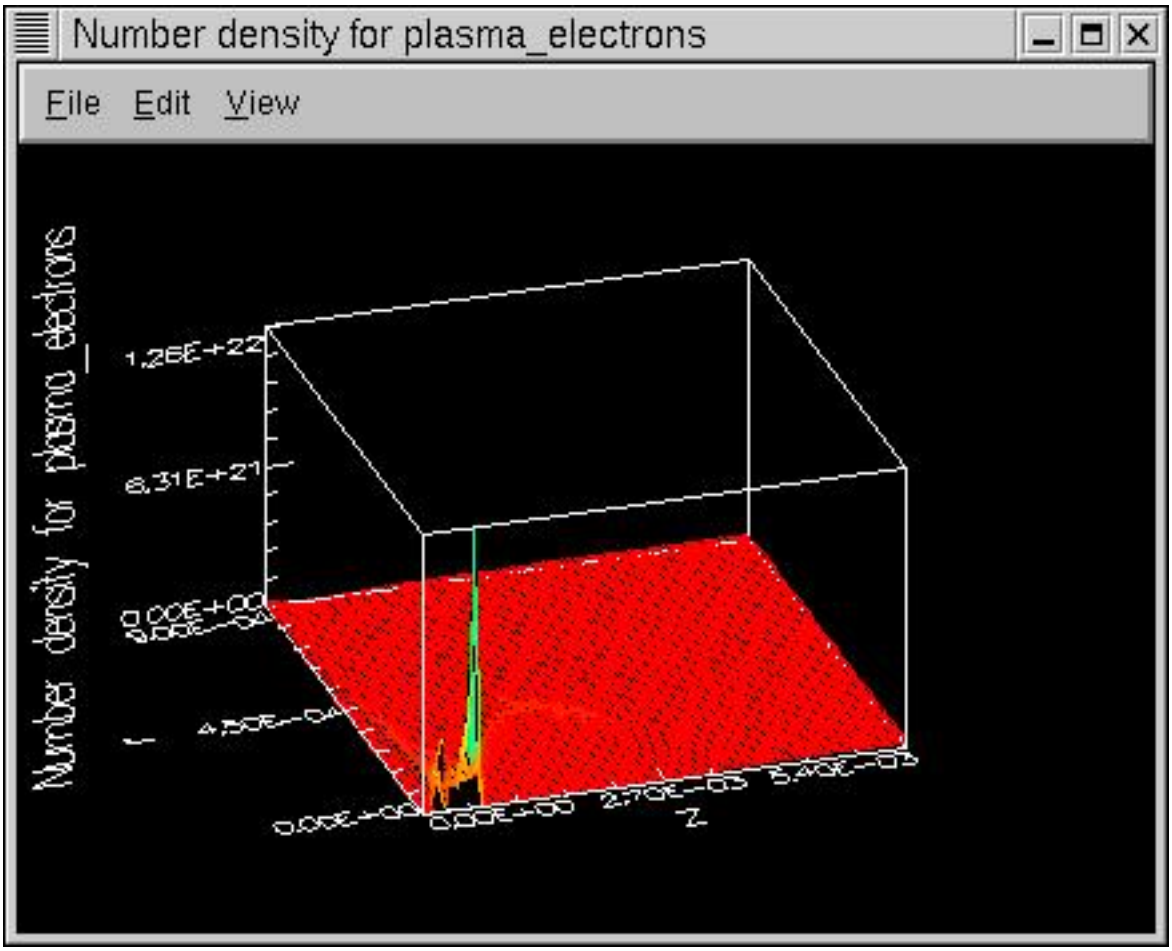

Figure 37: Plasma electron density spikes at the tail of the electron pulse.

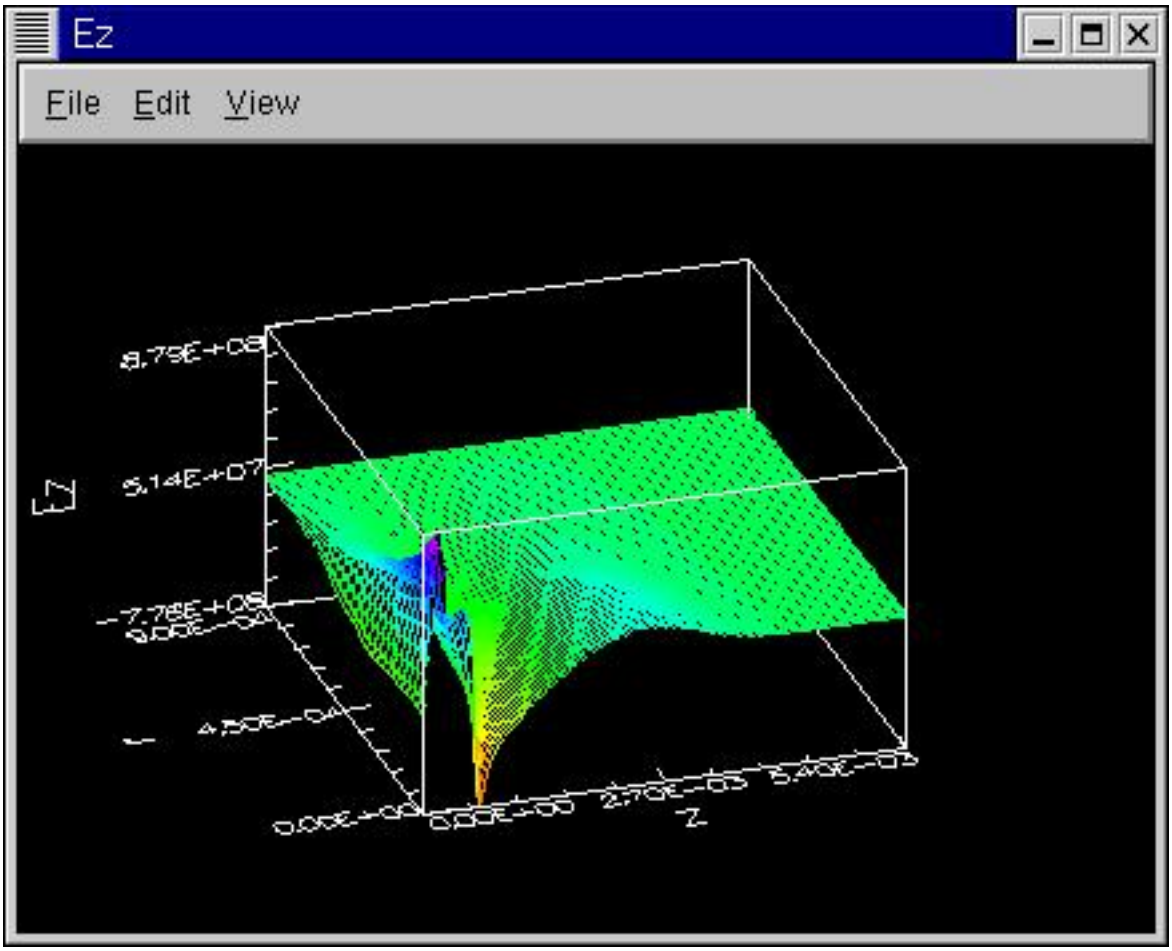

Figure 38: Electric field of electron pulse and surrounding plasma

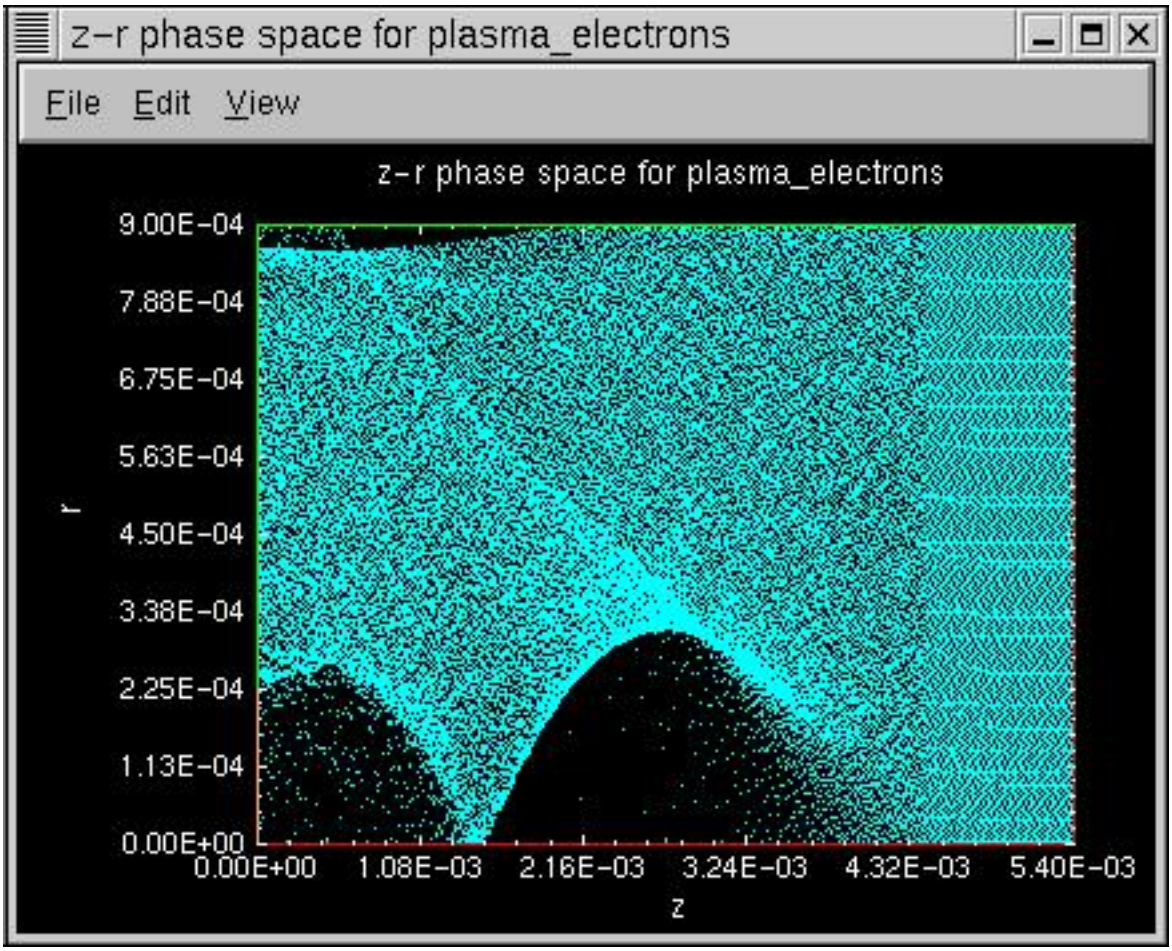

Figure 39: Wake in the plasma electron distribution for time $t = 2.00x10^{-11}$ sec.

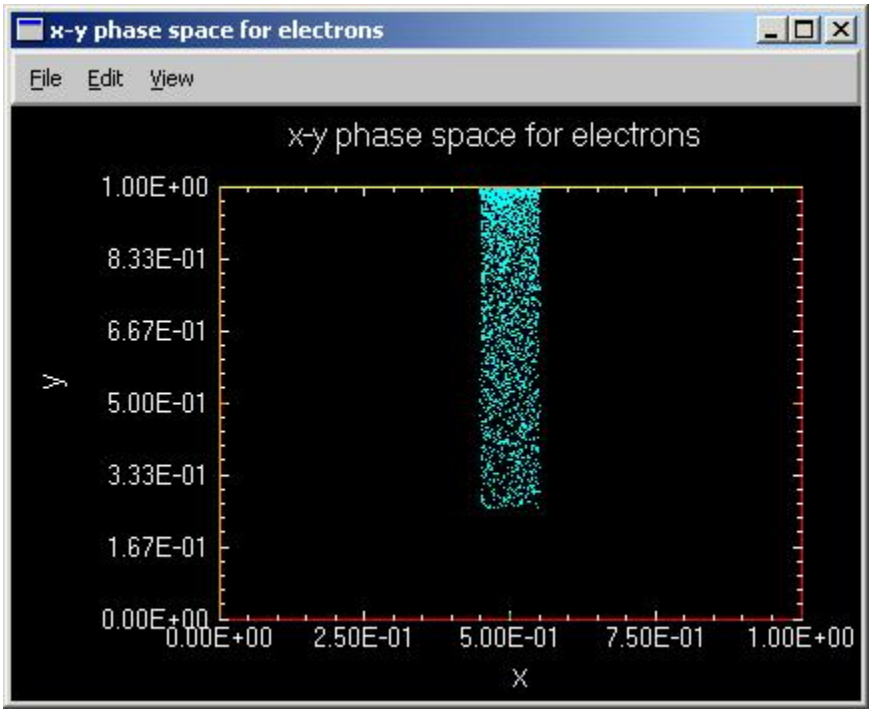

Figure 40: Electrons drift from the `EmitPort` element to the conductor across a 100V drop in potential

7.4 Toy Problem - Electrons dropping across a potential difference

Voltest.inp is a straightforward example of the use of OOPIC Pro to predict the energy change of electrons that are accelerated through a known potential. This example is illustrative of the use of OOPIC Pro diagnostics and the `EmitPort` *element. Other examples in this manual make use of the* `BeamEmitter` *element. The difference between the two lies in the application of conductor boundary conditions by the* `BeamEmitter` *element while* `EmitPort` *applies dielectric boundary conditions. In the OOPIC Pro online source code documentation,* `EmitPort` *can be seen to be a derived class of the* `BeamEmitter` *element.*

This example also illustrates the specification of special data accumulation diagnostics, which will measure the accumulation of electrons on a grounded, conducting boundary as well as the energy and placement of the incident electrons.

The geometric configuration of this example is expressed in Cartesian coordinates as a 1 m. by 1 m. grid divided into 20 cells along each direction. Since ElectrostaticFlag is set to 1, the DADI solver will be used to compute the EM fields that exist throughout the system. the timestep is set explicitly at 1 nanosecond.

The left and right boundaries of the system are defined to be dielectrics that will not be allowed to accumulate charge (QuseFlag = 0). This prevents any distortion of the electric field due to accumulated charge.

The top boundary of the system is defined to be an equipotential surface maintained at -100 V. An `EmitPort` *element is centrally located in this boundary. This will be the source of free electrons streaming in a current of 5 mA with a small, initial drift velocity corresponding to a low temperature.*

Figure 41 shows the drop in electric potential between the equipotential surface (maintained at -100V) and the bottom, conducting boundary. Limitations in the graphic system cause what seems, at first sight, to be a wide variation in potential across the device is actually just 1.6 V out of 100 V. There is also a "kink" in this rendition of the potential near the top of the device. If the resolution of the overlaid computational grid in the vertical coordinate is increased, it will be seen that the kink is actually evidence of oscillatory behavior that can be attributed to the proximity of the four boundaries of the system and the size of the `EmitPort` *being of the same order of magnitude as the extent of the boundaries. A revision of the input file to address these areas (i.e. increasing the ratio of horizontal extent of the*

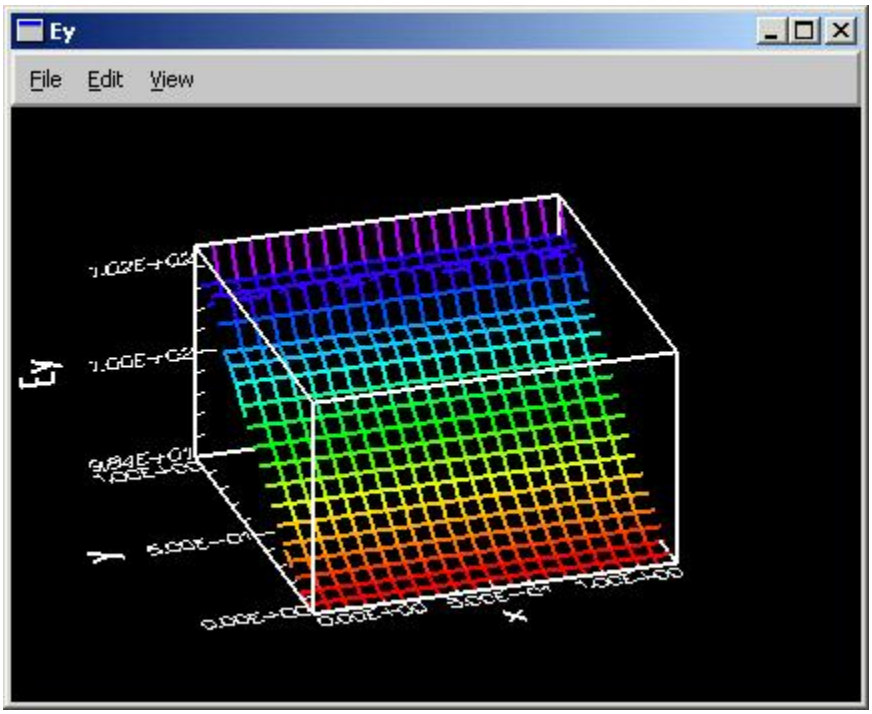

Figure 41: Electric potential drop between Emit Port on the top, equipotential surface and the bottom, conducting face

system to the width of the `EmitPort`*) can minimize the occurrence of oscillations.*

The bottom boundary is defined to be a `Conductor` *element. The diagnostics flag for this conductor is turned on (IdiagFlag = 1) so that statistics regarding the incidence of electrons at this face will be collected. A count of the number of incident electrons as well as the incident energy of the electrons is maintained. At the point in the simulation represented by Figure 40 neither of these diagnostics had started to accumulate due to the lack of incident electrons.*

Once the free electrons have had the time to traverse the gap between the `EmitPort` *and the conducting boundary, a count of the incident electrons can be collected as a function of position along the x axis combining two cells per bin for the data collection. As the simulation runs it can be seen that values of the count for each bin continues to increase but the important feature is the Gaussian shape of the distribution which indicates very little divergence of the electrons within the beam as shown in Figure 42.*

Figures 42 is a diagnostic plot made available by turning on the diagnostic flag IdiagFlag = 1 in the `Conductor` *element for the conducting boundary.*

7.5 Secondary particle production

This example ($OOPICPRO/input/gas.inp) illustrates the use of OOPIC Pro in modeling ionizing collisions between a streaming beam of electrons and a neutral gas contained in a chamber whose walls are conductors. The gas in this case is argon.

The input file for this example defines a 2D simulation, staged in cylindrical coordinates. z defines an axis of symmetry along which an electron beam is directed. Due to symmetry considerations, only one half of a slice through the z-axis needs to be modeled. The beam enters the chamber from its left face and propagates to the right. An OOPIC `BeamEmitter` *element is defined along part of this boundary, centered on the axis of symmetry, to launch the electron beam. The remaining boundaries of the simulation box are defined to be perfect conductors except for the z-axis, which is established by the CylndricalAxis element. The simulation boundaries represent one half of a single slice through the axis of symmetry, overlaid with a computational, two-dimensional grid.*

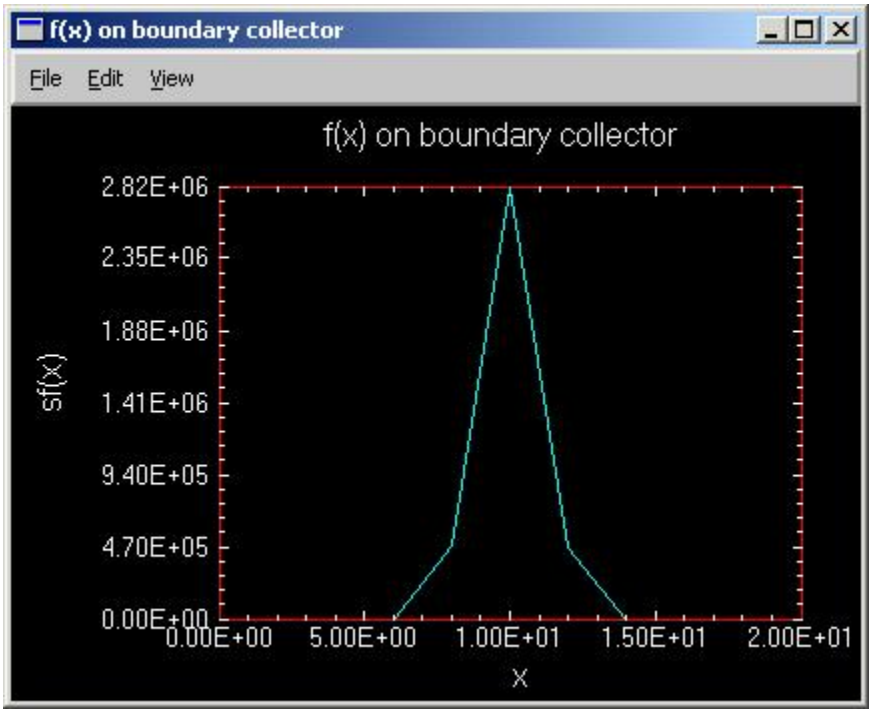

Figure 42: Gaussian shape of particle count at conducting boundary

The computational grid is defined to be 10 cells by 10 cells. This grid overlays the physical dimensions which measure 0.1 m horizontally and 0.02 m vertically so that there is a 5:1 aspect ratio for the modeled device. The example calls for the electron beam emitter element to have a radial extent on the boundary of five grid points, half of the left-most face of the chamber.

An interesting variation on this model will demonstrate the effect of the neutral argon gas. Setting the gas pressure to zero will effectively stream the electron beam into a vacuum. Electrons produced by the ionizing impact are no longer present to drift outside the electron beam. The divergence of the electron beam now occurs within a shorter distance of the `BeamEmitter` *source. The quicker divergence occurs because there are no longer any argon ions present to partially cancel the mutual repulsion among the electrons.*

8 References

1. J.P. Verboncoeur, A.B. Langdon and N.T. Gladd, Comp. Phys. Comm. 87, 199 (1995).

2. D.L. Bruhwiler, R.E. Giacone, J.R. Cary, J.P. Verboncoeur, P. Mardahl, E. Esarey, W.P. Leemans and B.A. Shadwick, "Particle-in-cell simulations of plasma accelerators and electron-neutral collisions," [142]Phys. Rev. Special Topics – Accel. & Beams, Issue 10 (October 2001).

9 Appendix A - Running OOPIC Pro in Batch Mode

This appendix contains information about using the batch processing capabilities of OOPIC Pro. The OOPIC Pro command line options will be explained. There will also be a brief introduction to some scripts useful for batch processing of data.

9.1 Command-Line Options

The command line can be used to start an OOPIC Pro run interactively with a GUI or in batch mode without a GUI. A simple example of running OOPIC Pro in batch mode from the command line is:

```
bin/oopicpro -i input/dring.inp -s 100 -nox
```

This command will run the dring (`-i input/dring.inp`) simulation for 100 steps (`-s 100`) without starting the GUI (`-nox`).

Table 9.1 is a list of the command line options and their descriptions.

Table 36: Command line options

Syntax	Default value	Explanation
`-i <input file>.inp`	No default. Specifying `-nox` without a valid `-i` argument results in an error	Specifies the name of the input file containing simulation parameters.
`-d<dump file>[.h5,.dmp]`	No default. Specifying `-d` without an argument results in an error. Default extension is .dmp. The extension is .h5 if and only if `-h5` is specified and `-or` is not specified.	Specifies the name of the file used for restoring restart data.
`-dp n`	-dp 0	Specify the number of iterations between dumps. `-dp 0` specifies no intermediate dump.
`-nox`	If `-nox` is not specified then run with X interface	If `-nox` is specified then run without X interface
`-s n`	`-s 0`	Specify the number of time iterations to be run. `-s 0` specifies no limit on iterations to be executed.
`-sf <save file>[.h5,.dmp]`	No default. Specifying `-sf` without an argument results in an error. Default extension is .dmp. The extension is .h5 if and only if `-h5` is specified and `-od` is not specified	Set the name of the save file. If both the '`-d`' and '`-sf`' options are set then '`-d`' sets the restore file, and '`-sf`' sets the save file. If only `-d` is set then the dumpfile has same filename stub as the restore file.
`-h5`	Dump and restore using HDF5 The `-h5` option with or without `-or` is ignored with regard to restoration of data if a .dmp extension is included in the specification of the restore file by the `-d` option.	Use HDF file format for dumping and restoring
`-h5 -or`	-h5 must be set	Restore from hdf5 format file
`-h5 -od`	`-h5` must be set	Dump to hdf5 format file
`-display displayname`	`-nox` must not be set. $DISPLAY is default device.	Specify the display device
`-exit`	Return to GUI control after completing iterations	Exit after completing iterations
`-h`		Display this list of options
`-p epsfile[.eps]`	No default. Specifying `-p` without an argument results in an error. The filename extension '.eps' is assumed if not specified.	Specify file name for postscript output.
`-u n`	`-u 1`	Update graphics displays every nth iteration.
`-id`	No incremental dumping.	Incremental dump files will contain a '_i' in the filename followed by the dump index.
`-dd`	No diagnostic data dump	Dump diagnostics corresponding to H5Diagnostic blocks in input file This option is valid only on Linux systems for which hdf5 file capability has been provided.

10 Appendix B - OOPIC Pro Postprocessing

This appendix presents information regarding the use of IDL (Interactive Data Language) Scripts to visualize data produced by OOPIC Pro.

10.1 Overview

A number of IDL (Interactive Data Language) scripts are provided with the OOPIC Pro distribution. They are useful for analyzing and visualizing the data produced by an OOPIC Pro simulation in a flexible, interactive or batch environment. In the IDL environment, data produced by OOPIC Pro can be visualized in a number of different formats. The scripts provided can be used to:

1. *Animate from one to four sequences of images created by saving individual plots (at the same simulation time point) as image files. Images from the selected sequences are combined according to their position in the sequence and displayed together allowing for convenient viewing and comparison of multiple diagnostic plots that change over time.*
2. *Plot the E Field of an OOPIC Pro simulation from an input text file created by dumping data from the E Field plot window for any single time point.*
3. *Plot a component of the E Field of an OOPIC Pro simulation from an input text file created by dumping data from the E Field component plot window for any single time point.*
4. *Recreate an OOPIC Pro particle phase plot from an input text file created by dumping data from a phase plot window for any single time point.*
5. *Recreate an OOPIC Pro particle density plot from an input text file created by dumping data from a density plot window for any single time point.*

Any of these plots can be created interactively or in a batch operation. The data (depending on its nature) can be visualized in a number of different formats:

1. *A 2-D line plot of data selected for a constant radial distance.*
2. *A 2-color (black and white) contour plot*
3. *A color contour plot*
4. *A surface plot*
5. *A combination 2-color (black/white) contour, color contour and surface plot displayed within a cube (IDL's Show3D)*

10.2 Interactive usage of IDL post-processing tools

The following sections describe the interactive use of IDL to post-process OOPIC Pro simulation data.

10.2.1 Data Animation

The OOPIC Pro IDL animation tool can only be used interactively since output from this tool is presented directly to a graphics display. To utilize the animation tool at least one group (time-ordered sequence) of image files must exist. The members of the file group are snapshots of a diagnostic variable captured at any number of discrete time points of an OOPIC Pro simulation. Use the manipulation techniques outlined in Section 4.6 to save image files (currently

only .bmp files are supported). To simplify usage of the animation tools the image files should be saved to the same directory location and their names should follow the format base_name.seq_no.bmp where:

base_name - is a descriptive filename that gives an indication of the image content

seq_no - is a numerical value indicating the position in the time series of images with the same base_name

bmp - is a filename extension that indicates that the file is stored in bitmap format

This is the same filenaming convention used by the OOPIC Pro movie creation facility as outlined in Section 4.8. It is much more convenient to use this method for the creation of the image files which will be collected into an animation sequence rather than to save plots individually at points where the simulation has been manually paused.

Run the IDL script `oopicpro_animate_new.pro` *from within the IDL development environment or from a UNIX command line prompt. IDL will prompt for the selection of the files to be included in the animation, one group (files that plot the same diagnostic parameter) at a time. The first dialog box prompting for the image files is labeled "Select first group of files to be animated". Multiple files are selected (from the same directory) using the shift-click method. If the file naming convention described above has been followed the filenames should appear sequentially in the file selection dialog box.*

Click on [open] to complete the file selection process. A subsequent dialog box appears that is labeled "Select second group of files to be animated". If the animation is to be produced for only one sequence of images then click [Cancel] to proceed to the animation. Otherwise select the same number of files for this second set as were selected for the first set and click [open].

Perform a similar sequence of selections for the third and (if necessary) fourth sequence of files. The IDL script will proceed directly to production of the animation without having to click [Cancel] if four file sequences are specified.

The IDL script resizes and mosaics the selected image sequences to fit the available space on the display device and produces the animation via IDL's XINTERANIMATE tool. For details on how to use XINTERANIMATE refer to the IDL Reference Guide (N-Z).

10.2.2 E Field plots

Text files containing E field profiles can be created during an OOPIC Pro simulation as outlined in Section 4.6. This data can be post-processed to produce images in a number of different formats either interactively or in batch mode with the IDL procedure `plot_efield_nogui.pro`*. This script can be executed in either an interactive or a batch mode. Invoke this procedure from within the IDLDE or from a UNIX command line.*

The application interface is divided into two primary areas. The area on the left of the application screen is topped by a set of usage instructions. According to the instructions an input file must first be opened by selecting the 'Open File' button at the top of the second interface section located on the right side of the window.

A dump file in text format created from the OOPIC Pro diagnostic plot of simulated E Field must be selected. Any other type of file will contain data in an incompatible format and will either produce unpredictable results, cause a runtime error or abort the IDL session.

Specification of an input file provides the IDL script with the base name of the data being processed. This IDL script will automatically create a '.sav' version of the data file if file input is successful read in. Subsequent sessions to process the same data can access this .sav file if the yes/no dialog box that follows the input file specification is answered 'yes'. Inputting data from an IDL .sav file is much faster than for text files for large datasets.

Labels and scale factors can be set to customize the presentation of the data in graphical form. Labels can contain any information desired. Scaling of data is turned on by setting the Scale Flag to 1 or 2. Otherwise no scaling is performed.

Setting the Scale Flag to 1 will scale coordinate data by a factor of $(c/w[0]) - 1$ *where c is the speed of light in cm/sec and* $iw[0]$ *is the laser frequency (for laser simulations). E field values are scaled by* $(m[e]/(c/w[0])) - 1$ *where* $m[e]$ *is the electron mass in eV.*

Setting the Scale Flag to 2 will scale coordinate data by a the laser wave vector whose magnitude is given by $k[p] = 2p * 9000.0 * sqrt(r)/c$ *where c is the speed of light in cm/sec. E field values are scaled by* $(m[e] * k[p]) - 1$.

Once labeling and scaling has been specified select any of the plot types listed and select [Plot] to produce a data plot in the Plot Preview Window. 'Lineplot' is used to plot a single slice of data for a constant radial position for all z. The slice is selected by moving the slider to specify a single vector of E field data from the available array. Values displayed by the slider correspond to the first and last row numbers within the two-dimensional array for E field data and are automatically determined from the data read from the input file.

Follow creation of the plot with selection of [Create PS] to create the same plot being currently viewed in postscript format.

Select the [done] button when post-processing has been completed.

10.2.3 E field component plots

Usage of `plot_Ezfield_nogui.pro` *is closely analogous to the description in 9.2.2 except that data must be reflective of a single component (e.g. the z component) of the E field rather than the full E Field. This script can be used in either an interactive or a batch mode.*

10.2.4 Particle density plots

Usage of `plot_ptcl_density_nogui.pro` *is also closely analogous to the description in 9.2.2 except that data must be reflective of data detailing particle density versus position. Allowance is made for the selection of different types of length scaling. The default scaling parameter is 1 which is in effect when 'None' is selected for the length scaling.*

10.2.5 Particle phase plots

The IDL script `plot_q_u_ptcl_nogui.pro` *will create plots of particle velocity as a function of position. The options available for plotting particle velocity versus position differ significantly from the other tools. Two plot types are available: Gamma and Energy. Gamma plots present particle velocities scaled by a factor* $g = sqrt(1 - v^2/c^2) - 1$ *versus particle position. Energy plots* $E_k = m_e((1 + ((gv)/c)^2 - 1)^{1/2}$ *versus particle position. Lengths in the plots can be scaled by either* c/w_0 *or plasma wavelength. Powers of 10 may be factored into or out of the values of position, r, or energy, E, by varying r_exponent or e_exponent respectively.*

10.3 Batch Usage of IDL post-processing tools

The same tools used to interactively visualize OOPIC Pro simulation results interactively can be invoked in batch mode as well (except for the animation tool). In batch usage, results are not rendered to a visual display but are instead saved as postcript files the same as those that would be created by selecting [Create PS] from an interactive session. This makes batch execution of the visualization tools as part of a large batch command stream possible for execution that will run unattended.

From a batch command stream, if IDL is invoked in the form:

```
IDL plot_type.pro, /NOGUI, cmdfile
```

IDL represents the platform-dependent method of invoking IDL. One of the IDL processing tools outlined in Section 10.2, as identified by type, will be invoked without an application interface. The IDL application will then read commands sequentially from cmdfile. The commands allowed in cmdfile are intended to mimic a subset of the steps taken in an interactive session. For instance, 'PLOT' is not allowed in batch mode since it would render a plot to application interface's plot window directly.

Each cmdfile should begin with an 'OPEN' 'data_file' command. Any commands to take action on data prior to its input via the OPEN command will cause the IDL session to abort.

10.3.1 Batch production of E Field plots

Batch processing and generation of postscript plots of OOPIC Pro simulation output is accomplished by entering a command such as the following (for UNIX/Linux):

```
IDL plot_Efield_nogui, /nogui, '$HOME/OOPIC_post_proc/idl_efld.in'
```

IDL will invoke IDL from a command line such as might be used on a UNIX system.

The contents of the file $HOME/OOPIC_post_proc/idl_efld.in might be:

```
'OPEN' '\$HOME/OOPIC\_post\_proc/IDL\_test\_files/efld\_ambipolar.txt'
'x1Label' 'ZZZZ'
'x2Label' 'RRRR'
'EcomponentLabel' 'EEEE'
'RADIO' '0'
'RADIO' '1' '35'
'RADIO' '2'
'RADIO' '3'
'RADIO' '4'
```

Each line of the command file issues an instruction to the IDL postprocessor application. The command names are reflective of the actions that might be used in the interactive environment. The sequence of commands given above would:

1. *open the E field data file saved from the OOPIC Pro simulation (paths to the data may be specified as an IDL environment variable or on the command line)*
2. *set the label of the x axis of all subsequent plots to 'ZZZZ' (plus unit labeling)*
3. *set the label of the y axis of all subsequent plots to 'RRRR' (plus unit labeling)*
4. *set the plot title of all subsequent plots to 'EEEE' plus appropriate information*
5. *produce a postcript file containing a surface plot of the E field versus position*
6. *produce a postcript file containing a line plot of the E field profile for the 35$\hat{}$th line of the E field array (array dimensions are set by the OOPIC Pro input file's `Control` block)*
7. *produce a postcript file containing a line contour plot of the E field versus position*
8. *produce a postcript file containing a color contour plot of the E field versus position*
9. *produce a postcript file containing plots on the face of a cube and within its volume combining a color contour plot, a surface plot and a line contour plot (IDL Show3D)*

The syntax for all commands requires the specification of parameters as quoted strings.

Additional supported commands are 'ScaleFlag' and 'ScaleInput' for turning scaling on or off and setting its value

For this input file a set of postscript files will created with the names:

efld_ambipolar_colorcon.ps
efld_ambipolar_contour.ps
efld_ambipolar_lineout.ps
efld_ambipolar_surface.ps

efld_ambipolar_tiered.ps

The name of each file combines a descriptor of the plot type with the input file name with an extension indicating that it is a postscript file.

10.3.2 Batch production of E Field component plots

Usage of the IDL script `plot_Ezfield_nogui.pro` *follows all of the guidelines that apply to* `plot_efield_nogui.pro`*. These two scripts utilize the same set of commands and the same syntax. The same command file shown above could be used for this script with a change to the name of the input file.*

10.3.3 Batch production of particle density plots

Use of `plot_ptcl_density_nogui.pro` *also follows use of the scripts for plotting an E field or an E field component. The command 'DensLabel' is substituted for 'EcomponentLabel' in the list of available commands. All other functionality remains the same.*

A sample command file would like:

```
'OPEN' '\$HOME/OOPIC\_post\_proc/IDL\_test\_files/ndens\_ions\_depstn.txt'
'x1Label' 'ZZZZ'
'x2Label' 'RRRR'
'DensLabel' 'EEEE'
'RADIO' '0'
'RADIO' '1' '70'
'RADIO' '2'
'RADIO' '3'
'RADIO' '4'
```

10.3.4 Batch production of phase plots

Batch execution of `plot_q_u_ptcl_nogui.pro` *accesses the same functionality as interactive execution and is similar to batch execution of the other IDL batch scripts with a different set of available commands.*

A sample command file would look like:

```
'OPEN' '$HOME/OOPIC_post_proc/IDL_test_files/qu_kly.txt'
'SCALING' '0'
'PLOT_TYPES' 'ENERGY'
'SCALING' '1'
'PLOT_TYPES' 'GAMMA'
```

Additional commands are:

'R_EXPONENT' 'value'
'E_EXPONENT' 'value'

11 Appendix C - Binary dump file format

This appendix contains information about the binary dump file format (dump files with '.dmp' extension). This dump file format predates the implementation of the HDF5 format and is still used for historic and support reasons.

11.1 Overview

The implementation of the binary dump file is somewhat non-intuitive. Objects such as the fields and grid are unique in the simulation so there is no difficulty in finding which object to investigate in the dump file. However, it is not clear how to find a particular boundary in the dump file. For this reason, all the boundaries (and diagnostics, which are analogous) have the same macroscopic format. There are a few data points of type int, one of which enumerates the number of data points to be loaded and another which identifies the type of boundary. Because of this uniform format, a boundary (or diagnostic) could be loaded without ever having a corresponding object in the simulation.

This method of formatting enables better support for multiprocessing. The multiprocessor version of OOPIC Pro geometrically divides the simulation region into one subdomain for each process. If dump and restore had been implemented by sending all the data to one process to do the all of the file I/O then dump and restore operations would have forfeited the advantage of multiprocessing. It is more reasonable to let each process handle it's own dump and restore operations. This avoids synchronization issues of many processes writing to the same file.

The files are named like this:

file.1 file.2 file.3 file.4.... file.n

for n processes created by `-np n`

The data is tagged with sufficient geometric data so that upon restore each process only needs to restore the parts of a particular dump file that belong in its portion of the simulation domain. If the dump file was created by a single process covering a single region that is then restored to two processes some of the boundaries might be cut in half. Each process will restore only the information from the data file that applies to it. Alternately, processes may look into several datafiles to gather all the data needed for its assigned domain, if the number of processes is reduced.

This is another reason for the uniform, predictable format of the boundary and diagnostic data dumps. A particular process might read a dumpfile and encounter a boundary not defined for it's domain. It will not to restore data for that boundary. Nevertheless, it must proceed to the next boundary. With a uniform format, the process will know how to skip to the next boundary.

11.2 Details

Each binary dump file begins with:

Table 37:

Data	Purpose	Where	Example
4 char	Version code	xg/dump.cpp	2.61
1 int	random number seed	physics/sptlrgn.cpp	43498593
1 int	length of simulation name	physics/sptlrgn.cpp	10
n+1 char	the chars of the name	physics/sptlrgn.cpp	"electron gun"
1 float	simulation time	physics/sptlrgn.cpp	1.233e-9
4 float	Physical region covered	physics/sptlrgn.cpp	xs,ys,xf,yf

11.2.1 Boundary data

Generic boundary data *This block provides generic boundary data. This block accompanied by specific boundary data is repeated nbound times, once for each boundary in the simulation as specified in the preceding block.*

Table 38: Boundary data format

Data	Purpose	Where	Example
1 int	number of boundaries, nbound	physics/spltrgn.cpp	20

Table 39: Generic boundary data

Data	*Purpose*	*Where*	*Example*
4 float	*dimension descriptor*	*physics/boundary.cpp*	*Xs,ys,xf,yf*
1 int	*type of this boundary*	*physics/boundary.cpp*	*3*

Include descriptors for boundaries (see examples below).

Variable Boundary Data *Example formats are given here for* `ExitPort`, `Dielectric` *and* `DielectricRegion` *boundaries. This information completes the boundary specification along with the generic data above. These specifications must ALL follow the format: 1 int (n), 1 int (unused), 1 int(unused), 4n floats*

This is the format for an `ExitPort` *(accompanies generic block above):*

Table 40: `ExitPort` *data format*

Data	*Purpose*	*Where*	*Example*
1 int	*number of datapoints, n*	*Physics/boundary.cpp*	*5*
1 int	*(unused)*		
1 int	*(unused)*		
4n floats	*x,y,oldH1,oldH2 (the x location, the y location (in MKS) and H stored there)*	*Physics/boundary.cpp*	

This is the format for a `Dielectric` *and* `DielectricRegion` *(accompanies generic block above):*

Table 41: `Dielectric` *and* `DielectricRegion` *format*

Data	*Purpose*	*Where*	*Example*
1 int	*number of datapoints, n*	*physics/dielectr.cpp*	*7*
1 int	*(unused)*		
1 int	*(unused)*		
4n float	*x,y,Q(x,y), unused*	*physics/dielectr.cpp*	*x,y,Q(x,y),0*

11.2.2 Diagnostic data

Data describing the format of diagnostic data follows the boundary definitions.

Table 42: Diagnostic data format

Data	*Purpose*	*Where*	*Example*
1 int	*number of diagnostics, ndiag*	*physics/spltrgn.cpp*	*4*

Generic diagnostic data, first block *A block of generic diagnostic data, coupled with the diagnostic data block is repeated ndiag times, once for each diagnostic.*

Table 43: Generic data, first block format

Data	*Purpose*	*Where*	*Example*
4 float	*dimension descriptor*	*xg/newdiag.cpp*	*xs,ys,xf,yf*
1 int	*xlength*		
1 int	*ylength*		
1 int	*title length*	*xg/newdiag.cpp*	*10*

1 int	*history flag*	*xg/newdiag.cpp*	*0*

Include second block of descriptors for diagnostics (see examples below).

Generic diagnostic data, second block *Data descriptors all follow the format 3 ints, hist_num floats*

Table 44: Generic diagnostic data, second block format

Data	*Purpose*	*Where*	*Example*
1 int	*hist_num*	*xg/history.cpp*	*3*
1 int	*hist_hi*	*xg/history.cpp*	*3*
1 int	*alloc_size*	*xg/history.cpp*	*3*
hist_num floats	*the time array*	*xg/history.cpp*	

Additional data specific to diagnostic.

Format is: 1 int (num_hist), 11 more ints, num_hist floats

Table 45: Definition of a Scalar_History diagnostic

Data	*Purpose*	*Where*	*Example*
1 int	*hist_num*	*Xg/history.cpp*	*3*
1 int	*Comb*	*Xg/history.cpp*	*3*
1 int	*n_comb*	*Xg/history.cpp*	*0*
1 int	*take_state*	*Xg/history.cpp*	
1 int	*Step*	*Xg/history.cpp*	
7 ints	*Unused*		
hist_num floats	*the data array*	*Xg/history.cpp*	

Table 46: Definition of a Scalar_Local_History diagnostic

Data	*Purpose*	*Where*	*Example*
1 int	*hist_num*		
1 int	*left_shift*	*Xg/history.cpp*	*5*
10 ints	*Unused*		
hist_num floats	*the data array*		

Table 47: Definition of a Scalar_Local_History diagnostic

Data	*Purpose*	*Where*	*Example*
1 int	*hist_num*	*xg/history.cpp*	*5*
1 int	*left_shift*	*xg/history.cpp*	*5*
1 int	*Ave*	*xg/history.cpp*	*5*
1 int	*Step*	*xg/history.cpp*	*5*
1 int	*take_state*	*xg/history.cpp*	
7 ints	*Unused*		
hist_num floats	*the data array*		

Table 48: Definition of a Scalar_Ave_History diagnostic

Data	*Purpose*	*Where*	*Example*
1 int	*hist_num*		*1*
1 int	*left_shift*		*2*
1 int	*Ave*		*3*
1 int	*step_ave*		*4*
1 int	*take_state_ave*		*5*
1 int	*Comb*		*6*
1 int	*step comb*		*7*
1 int	*N_comb*		*8*
1 int	*take_state_comb*		*9*
1 int	*Step*		*10*
2 ints	*Unused*		*11 & 12*
hist_num floats	*the data array*		

Table 49: Definition of a Vec_Pointers_History diagnostic

Data	*Purpose*
1 int	*datasize = hist_num*vector_size*
1 int	*Comb*
1 int	*N_comb*
1 int	*take_state*
1 int	*Step*
1 int	*vector_size*
6 int	*Unused*
*hist_num * vector_size floats*	*the data*

Table 50: Definition of a Vec_Pointers_History_Ave diagnostic

Data	*Purpose*
1 int	*datasize = hist_num*vector_size*
1 int	*Comb*
1 int	*Ave*
1 int	*N_comb*
1 int	*take_state*
1 int	*Step*
1 int	*vector_size*
5 int	*Unused*
*hist_num * vector_size floats*	*The data*

Table 51: Definition of a Vec_Pointers_History_Local_Ave diagnostic

Data	*Purpose*
1 int	*datasize = hist_num*vector_size*
1 int	*left shift*
1 int	*Ave*
1 int	*take_state*
1 int	*Step*
1 int	*vector_size*
6 int	*Unused*

*hist_num * vector_size floats*	*the data*

Table 52: Definition of a Vector_History diagnostic

Data	*Purpose*
1 int	*datasize = hist_num*vector_size*
1 int	*comb*
1 int	*N_comb*
1 int	*take_state*
1 int	*step*
1 int	*vector_size*
6 int	*unused*
*hist_num * vector_size floats*	*the data*

Table 53: Definition of a Vec_Pointers_Local_History diagnostic

Data	*Purpose*
1 int	*datasize = hist_num*vector_size*
1 int	*left shift*
1 int	*vector_size*
9 int	*unused*
*hist_num * vector_size floats*	*the data*

Table 54: Definition of a Vector_Local_History diagnostic

Data	*Purpose*
1 int	*datasize = hist_num*vector_size*
1 int	*left shift*
1 int	*vector_size*
9 int	*unused*
*hist_num * vector_size floats*	*the data*

Table 55: Definition of a JE_Region_History diagnostic

Data	*Purpose*
1 int	*datasize = x*y*
1 int	*ave*
1 int	*X_array_size*
1 int	*Y_array_size*
8 int	*unused*
datasize floats	*the data*

Table 56: Definition of a Region_History diagnostic

Data	*Purpose*
1 int	*datasize = x*y*
1 int	*hist_hi*
1 int	*ave*
1 int	*X_array_size*
1 int	*Y_array_size*

7 int	*unused*
datasize floats	*the data*

Table 57: Fields format

Data	*Purpose*
1 int	*J*
1 int	*K*

Diagnostic-specific data *Block to be repeated J*K times:*

*Table 58: Repeated Block (J * K times)*

Data	*Purpose*
1 int	*X location of these fields*
1 int	*Y location of these fields*
1 vector3	*intEdl(x,y)*
1 vector3	*intBds(x,y)*
1 vector3	*I(x,y)*

Table 59:

Data	*Purpose*
1 float	*EMdamping*

*If EMDAMPING $>$ 0 then repeat this block J*K times*

Table 60: Repeated block if EMDAMPING $>$ 0

Data	*Purpose*
1 float	*xloc*
1 float	*yloc*
1 Vector3	*intEdlBar(x,y)*

Table 61: Particles format

Data	*Purpose*
1 int	*SpeciesID*
1 flag	*vary_np2c*
1 float	*np2c0*
1 int	*N*

If (vary_np2c) then repeat this block n times:

Table 62: Block repeated if vary_np2c

Data	*Purpose*
1 float	*qarray(i)*
1 vector2	*X*
1 vector3	*u*

else repeat this block n times

Table 63: Repeat this block if vary_np2c not defined

Data	*Purpose*
1 vector2	*X*
1 vector3	*u*