VSim User Guide

Release 11.0.1-r3016

Tech-X Corporation

Jun 29, 2021

CONTENTS

OVERVIEW

This manual demonstrates how to use either the Visual Setup (.sdf input files) or Text Base Setup (.pre input files) to set up a VSim simulation. It then shows how to run the computational engine on the resulting input files, how to perform data analysis in VSim, and how to visualize data.

VSim [VSi] is an arbitrary dimensional, electromagnetics and plasma simulation code consisting of two major components:

- VSimComposer, the graphical user interface.
- Vorpal [NC04], the VSim Computational Engine.

VSim also includes many more items such as Python, MPI, data analyzers, and a set of input simplifying macros.

1.1 What is VSimComposer?

VSimComposer provides an interface that allows you to edit and validate your simulation input files, run VSim simulations, and visualize results using the VisIt-based Visualization pane embedded within VSimComposer. You can also edit VSim input files in the text editor of your choice, perform calculations with the easy-to-run, command-line-driven Vorpal engine, and then run the visualization tool of your choice.

Note: Look and Feel Differences

VSim runs on Linux, Mac OS X, and Windows. Each of these platforms has its own unique look and feel (e.g. the ordering of buttons in dialogs, and in the case of Mac OS X the arrangement of menus and menu items). Furthermore, the appearance of VSim can vary depending on the theme being used.

Given these differences and that screenshots may not always be of the most recent release, please note that screenshots shown in this manual may look different from the VSim that you see running on your own computers. VSim should function the same, but if you look for a particular toolbar or button, it may not be in same place in your copy of VSim.

1.2 VSim Capabilities

VSim is a flexible, multiplatform, high-performance tool for running electromagnetic, electrostatic, and plasma simulations in 1, 2, or 3 dimensions.

VSim solves EM propagation in the presence of complex dielectric and metallic shapes with accurate simulation of curved geometries using a conformal mesh. Examples include:

- Dish Antenna
- Horn Antenna

- Patch Antenna

- Waveguides

- Far Field calculations

- Radar Cross Sections

- Crab Cavities

The kinetic plasma model is based on the particle-in-cell (PIC) algorithm, both in the electromagnetic and electrostatic limits. For electromagnetics, a charge-conserving current deposition algorithm enables the integration of the Maxwell equations without any additional divergence correction. In the electrostatic limit, the VSim computational engine (Vorpal) solves Poisson's equation at every time step based on the instantaneous charge distribution. Examples include:

- Magnetrons

- Electron guns

- Ion sources

- Multipacting in waveguides

- Traveling Wave Tubes

- Hall Thrusters

- Laser Plasma Accelerators

The structures within the VSim applications can be arbitrarily shaped and can define the locations of conductors, dielectrics, particle absorbers, reflectors, and other geometrical objects.

TWO

STARTING VSIMCOMPOSER

2.1 Running Locally

Once you have installed VSim successfully as per the instructions given by VSim Installation, you are ready to run the program. This section will detail how to run VSim locally on Windows, Mac, and Linux operating systems.

2.1.1 Running VSim on Windows

You can start VSim in the following ways on Windows:

- Select the VSim Icon
    - Double-click on the VSim shortcut on your desktop
    - Go to your Start menu, navigate to the Tech-X folder, and click on the VSim icon
- Run VSim from the Windows Command Line
    - Navigate to the relevant program folder by going to *Program Files -> Tech-X -> VSim-10.0 -> Contents -> bin*
    - Type in *VSimComposer.exe* to run the VSim executable and launch the program
- Open VSim through Windows File Explorer
    - Open Windows File Explorer, either through your start menu, desktop shortcut, or taskbar icon
    - Navigate to C:Program FilesTech-XVSim-10.0
        * You can either click on the *VSimComposer.lnk* shortcut in this folder, or. . .
        * Continue on to *Contents –> bin* and then double-click on *VSimComposer.exe*
- Open VSim using an existing file
    - Open Windows File Explorer
    - Select a file that has a VSim-supported extension (like a .pre, .in, or .sdf, for example). Either. . .
        * Double-click on the file if its default program is VSimComposer
        * If its default program is not VSimComposer, right-click on the file and select **Open with**. Go into *Program Files -> Tech-X -> VSim-10.0 -> Contents –> bin* and select the *VSimComposer.exe* application file.

2.1.2 Running VSim on Mac

On Mac OS, the methods for starting VSim are:

- Start from the Applications Folder

    - Click on the VSimComposer icon in the Applications/VSim-10.0 folder (or in the folder where you have VSim installed)

- Run VSim from Terminal Window

    - Open a Terminal window

    - Navigate to the folder where VSimComposer is installed, most likely by going to */Applications/VSim-10.0/VSimComposer.app/Contents/MacOS*

    - Start VSim by typing *./VSimComposer.sh* to run the program executable

2.1.3 Running VSim on Linux

For Linux, you can start VSim through the following:

- Navigate to the folder where the program is installed, for example */user/local/VSim-10.0-Linux64*

- Type in *./VSimComposer.sh* to run the program executable

2.2 Running VSimComposer On a Remote Computer System

Just as on a local workstation or laptop, the computational engine (Vorpal) may be invoked through the graphical interface or from the command line on a remote system. On high performance computing clusters the command line approach may be required in order to submit a job to a resource management system. These are documented separtely here: - *Running Vorpal from the Command Line* - *Running Vorpal from a Queueing System*

In this section we discuss alternatives for setting up, running via the GUI, and visualising output.

In the present version we offer the following capabilities for running VSim remotely:

Note: Prior to starting up VSim, it may be necessary to set the environment variable `export LIBGL_ALWAYS_INDIRECT=1` in order for the visualization stage to work correctly. Some users using VNC on ubuntu 14.04 have also reported adding the system installation of mesa to the start of the LD_LIBRARY_PATH environment variable, has helped overcome their issues.

`export LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu/mesa:$LD_LIBRARY_PATH`

- Remote Desktop: We have tested two different methods for setting up a remote desktop on a server. One method uses VNC such as Tight VNC and Turbo VNC. If a VNC server is set up on the remote machine, one may try to connect a local client to the remote machine using VNC. Another method is through the use of NoMachine. In order to log in to a remote server, the same software needs to be installed on both the server and the client. Please work with your local support staff to ensure any one of these software solutions is correctly installed on both the server and client.

- Virtual GL: If one is using a remote machine that has virtualGL server or DCV one may run using the hardware acceleration on the remote machine. This may provide the best performance but also the most system administration work. Many HPC centers are already set up for this kind of access. There is a super-accelerated virtualGL client, but it is more common to find virtualGL set up like DCV such that the remote machine is running a VNC server, which you may connect to with any VNC client on your local machine.

- X Windows: If one does not have hardware acceleration on the remote machine one may forward X using an ssh client (`ssh -Y`) or use accelerated X forwarding using software like NoMachine NX. As of this writing, a good discussion is at (https://www.hoffman2.idre.ucla.edu/access/x11_forwarding). Briefly,

    - Linux users running X: edit /usr/bin/Xorg as described at the above link.

    - OS X users running XQuartz: execute

        ```
        defaults write org.macosforge.xquartz.X11 enable_iglx -bool true
        ```

    - Windows users: many options described at the above link. In addition, you can install an X server (such as XMing) on your Windows machine. You will also need to install Cygwin. Follow these steps to open Composer on a remote Linux machine. (1) Start the XMing server on your local computer. (2) Open Cygwin and type the following commands in the Cygwin terminal: - startxwin & - export DISPLAY=:0.0 - xterm & - ssh -XY address.to.your.remote.machine - navigate to location of installed VSim software and issue the command ./VSimComposer.sh

        If X11 forwarding is enable on the remote machine, then this should open a Composer window.

2.3 Visualizing Remote Data

The following capabilities are recommended for visualizing remote data if the previous recommendataions do not work for you:

- One may use an external utility to copy the remote files back to a local machine (scp, sftp or winSCP are likely options). Providing the .pre or .sdf file is in the same directory as the data to be viewed, it should be possible to visualize the output locally. This is the most appropriate solution for those dealing with small datasets.

- Use standalone VisIt and it's remoting feature. VisIt may be downloaded from (https://wci.llnl.gov/simulation/computer-codes/visit/downloads). Users of native VisIt should be sure to make sure their local version number matches up with the version running remotely. This option offers the full flexibility of VisIt, such as the ability to make all the individual images needed to make movies in a single go and the ability to fully Python script your visualisation, but there is a corresponding learning curve. There are many tutorials and resources at The visitusers website (http://www.visitusers.org).

- Use matplotlib or alternative software on the remote machine. The VsHdf5 module is provided with VSim and may be used to read and manipulate remote datasets for this purpose.

- If you are using VNC or NoMachine, you can open any data set through Composer just as you would on your own desktop.

2.4 Welcome Window

Upon opening VSim, you are brought to the Welcome Window. If it is your first time opening VSim, your *Recent Simulations* will be empty. However, if you have completed previous runs, you may use this area to quickly select a recent simulation and re-open it. You can also create new simulations based on those you have recently worked with. The Welcome Window is shown in Fig. 2.1.

Fig. 2.1: Welcome Window

CREATING OR OPENING A SIMULATION

3.1 Starting a Simulation

The first step to creating your own simulation is to open a file. It can be an existing example, an existing simulation, or a completely new simulation.

3.1.1 Creating a Simulation from an Example

To run one of the examples in VSim, one must first copy the example file, and any correlated files, to a new directory. Choose *New -> From Example* from the **File** menu. See Fig. 3.1.
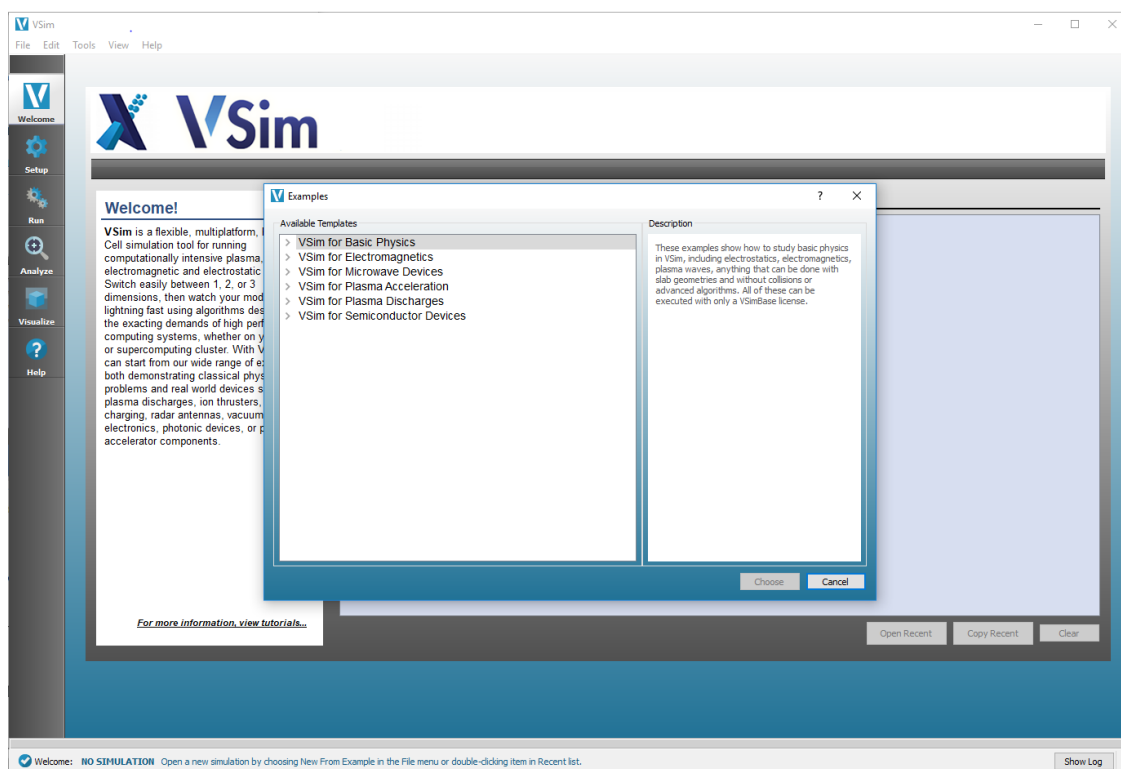
Fig. 3.1: Create new simulation from example

Select Example Template

The *Examples* dialog box will open and you can choose an example template to run. Here a short description and image of the available examples will be displayed. Examples are split into *VSim Modules*. You must possess the correct license for an example to run, though you can see them all.

Upon selecting an example from the *Available Templates* pane, either double click the example you have chosen or single click and then click the *Choose* button.

For this demonstration we have chosen the *Electromagnetic Plane Wave* example from the *VSim for Basic Physics* module. See Fig. 3.2.
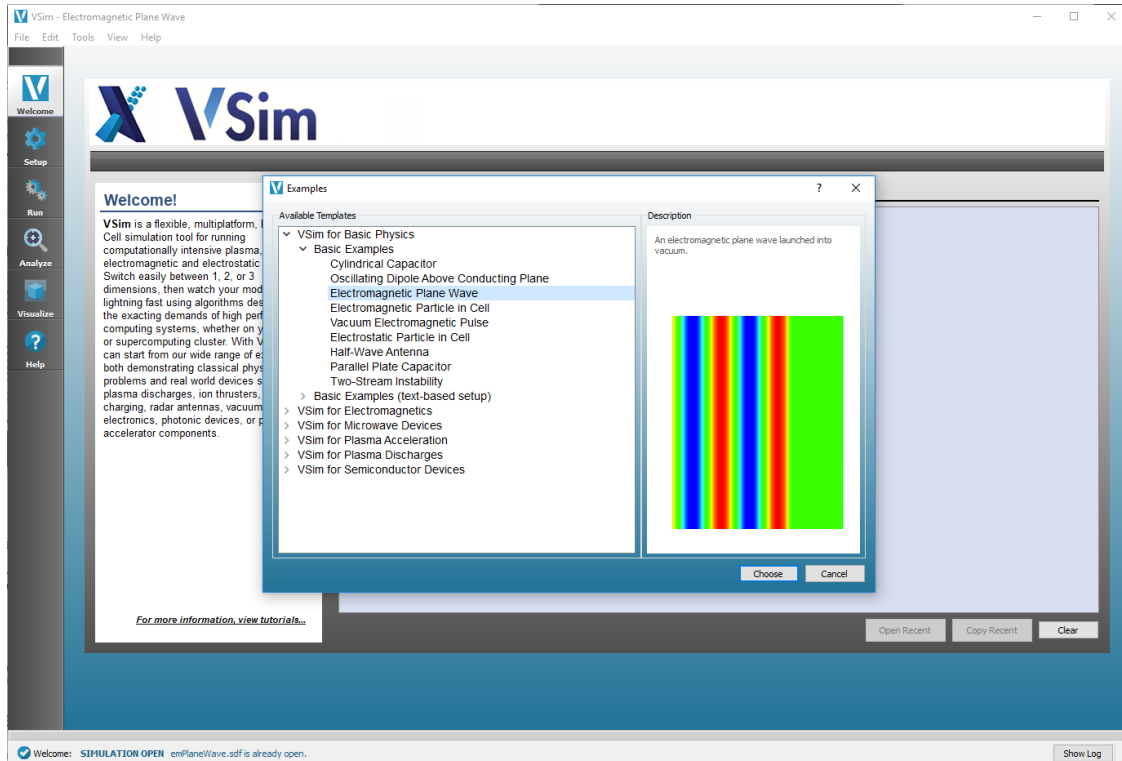
Fig. 3.2: Selecting an example from the *Examples* dialog

Choose a Name for the New Simulation

Upon selection of the example template, the window *Choose a Simulation Name* will open. Here you can choose the folder, or create a new folder, and set the name of the new simulation.

When the name and directory have both been chosen, click the *Save* button to proceed. See Fig. 3.3.

Proceed to the Setup Window

From here, Composer will automatically take you to the **Setup** window where you are free to explore and alter the parameters of the example you have chosen. See Fig. 3.4.
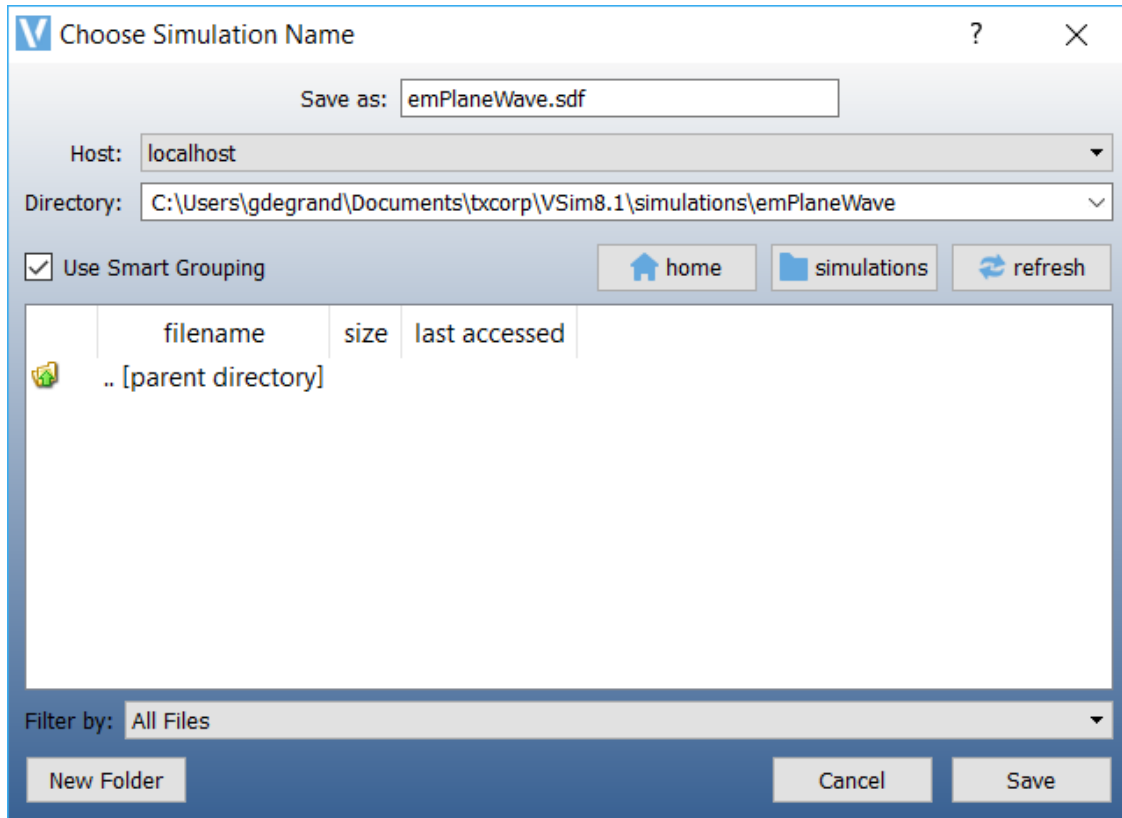
Fig. 3.3: Choose name and directory to use for the simulation
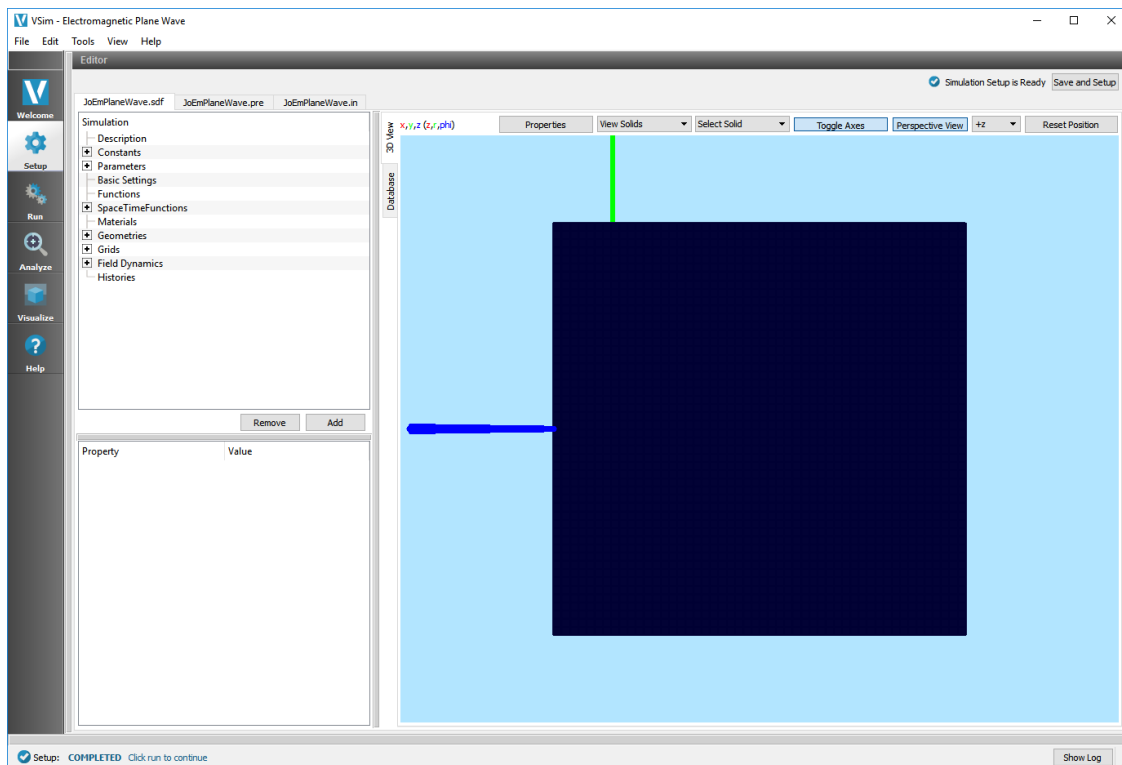
Fig. 3.4: Proceed to the setup window

3.1.2 Opening an Existing Simulation

If you have a simulation already created, you can double-click on the simulation .sdf or .pre file that appears under *Recent Simulations* in the VSim Composer **Welcome** menu.

Or, if your simulation does not show under the *Recent Simulations* heading, you can go to *File -> Open Simulation* and use the *Open Simulation* file navigation window to select the simulation .sdf or .pre file you would like to open. See Fig. 3.5.
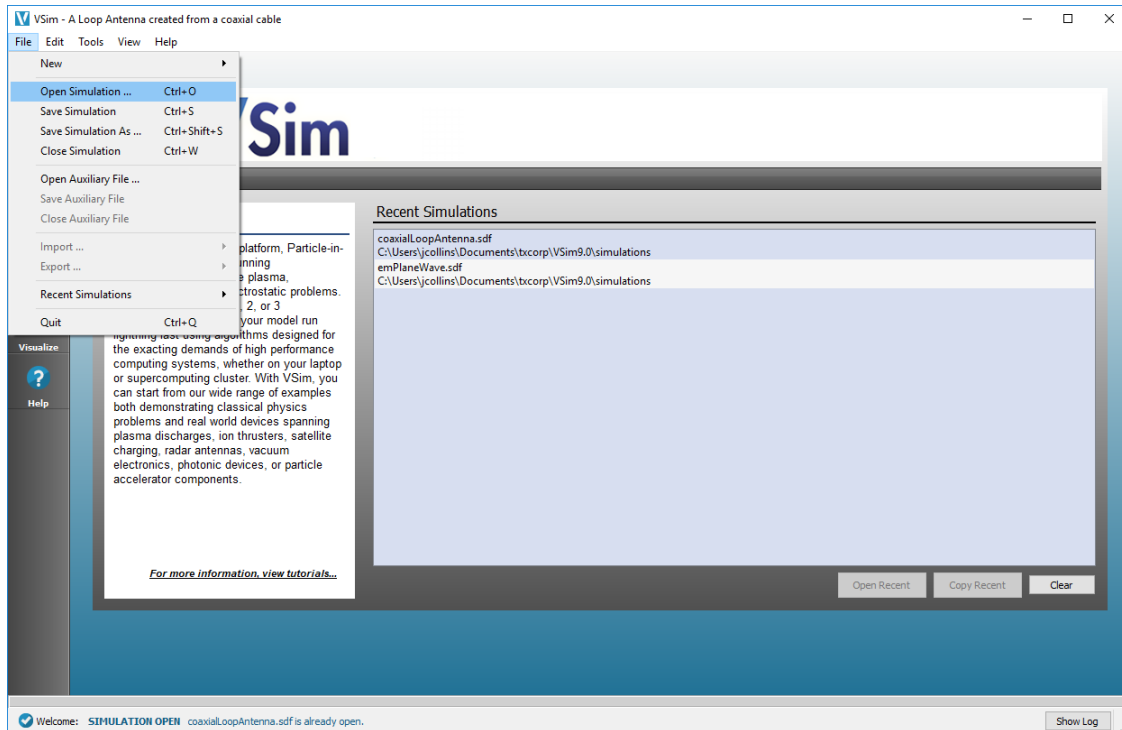
Fig. 3.5: Open an existing simulation .sdf or .pre file

From here, Composer will automatically take you to the **Setup** window where you are free to explore and alter the parameters of your simulation.

3.1.3 Creating a Blank Simulation

Create New Visual Setup Simulation

To create a completely custom visual-based simulation, you must choose *New -> Simulation* from the **File** menu. Creating an entirely new simulation requires knowledge of topics covered in later sections. See Fig. 3.6.

Proceed to the Setup Window

You are automatically taken to the **Setup** window. You are now free to create your own custom simulation and explore the power and versatility of the Vorpal engine. See Fig. 3.7.
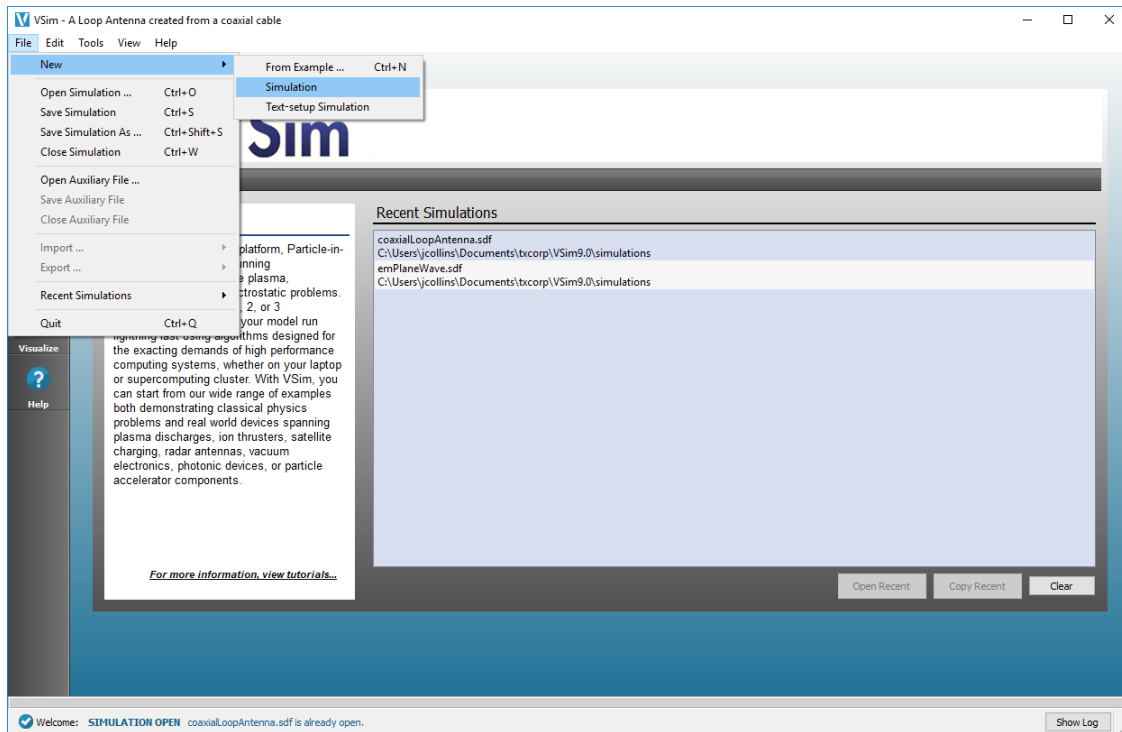
Fig. 3.6: Choose to create a new visual setup simulation
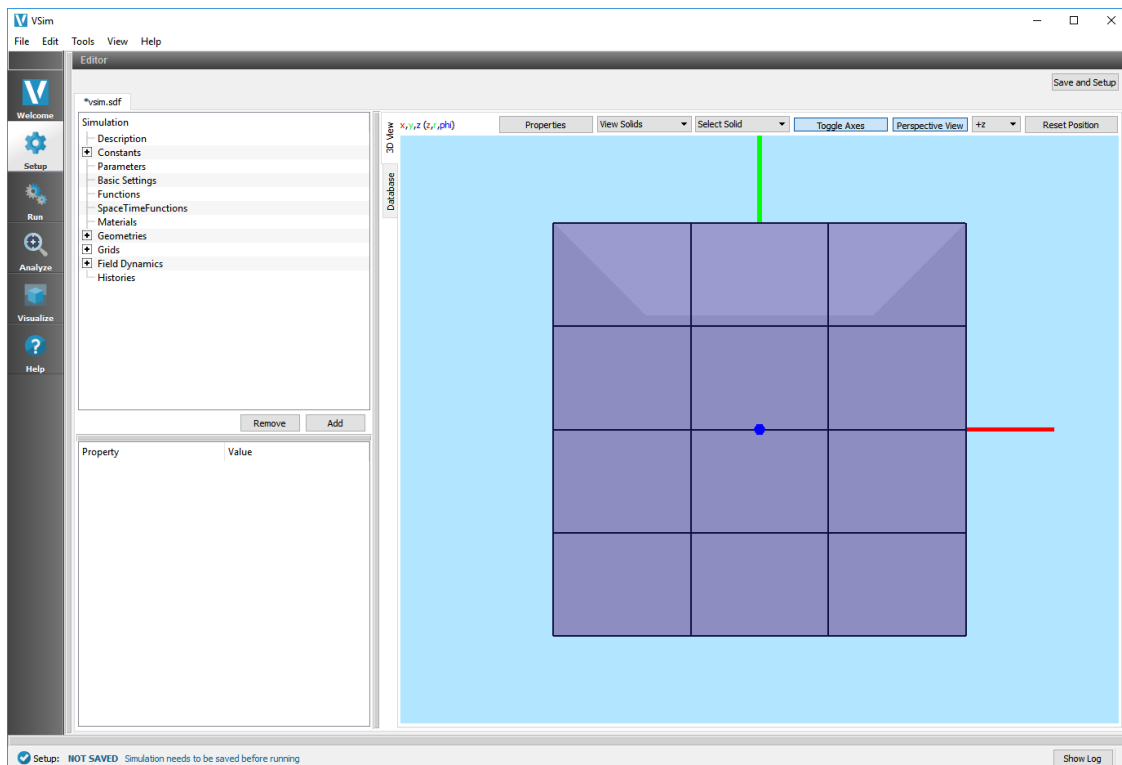
Fig. 3.7: A blank slate

Choose Simulation Name and Save

When you are ready to save your simulation, go to *File -> Save Simulation* and choose a location and name for your new simulation. See Fig. 3.8.
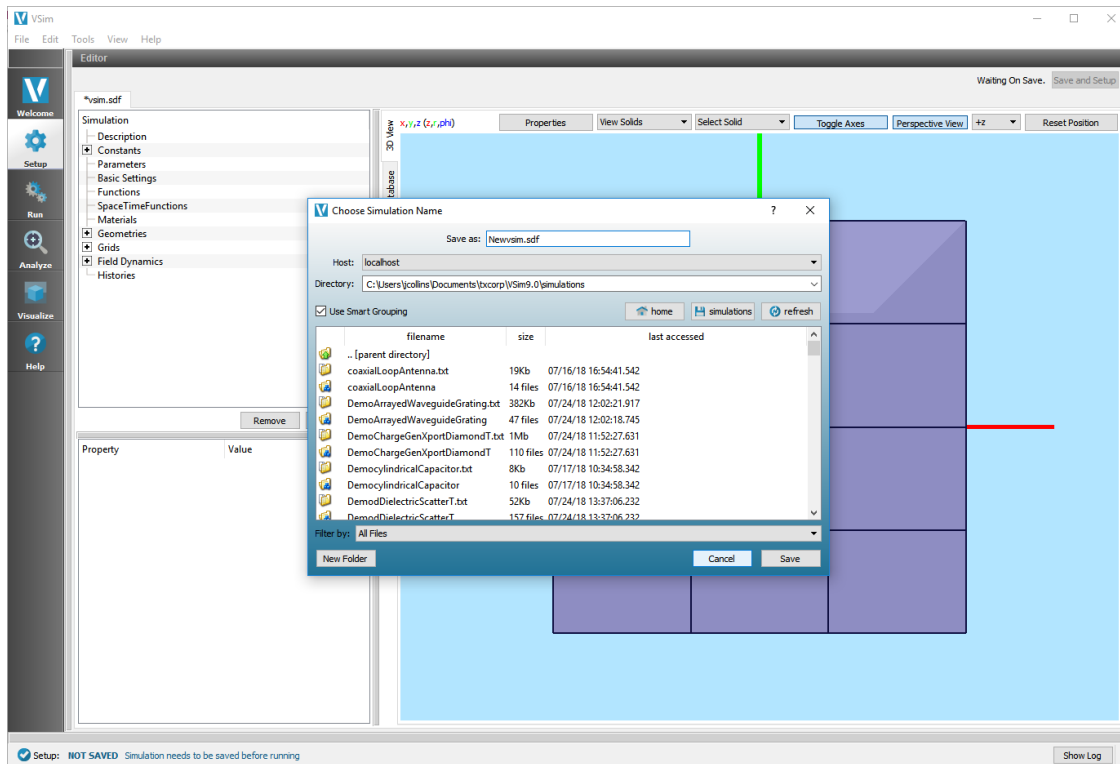
Fig. 3.8: Choose a name and directory for your new visual setup simulation

3.1.4 Create New Text Setup Simulation

To create a completely custom text based simulation, you must choose *New -> Text-setup Simulation* from the **File** menu. Creating an entirely new simulation requires knowledge of topics covered in later sections. See Fig. 3.9.

Choose New Simulation Name And Save

A new window entitled *Choose a name for the new simulation* will pop up. Here, you choose a name for your new simulation along with a location where the simulation will be saved. See Fig. 3.10.

Proceed to the Setup Window

You are automatically taken to the **Setup** window. You are now free to create your own custom simulation and explore the power and versatility of the Vorpal engine. See Fig. 3.11.
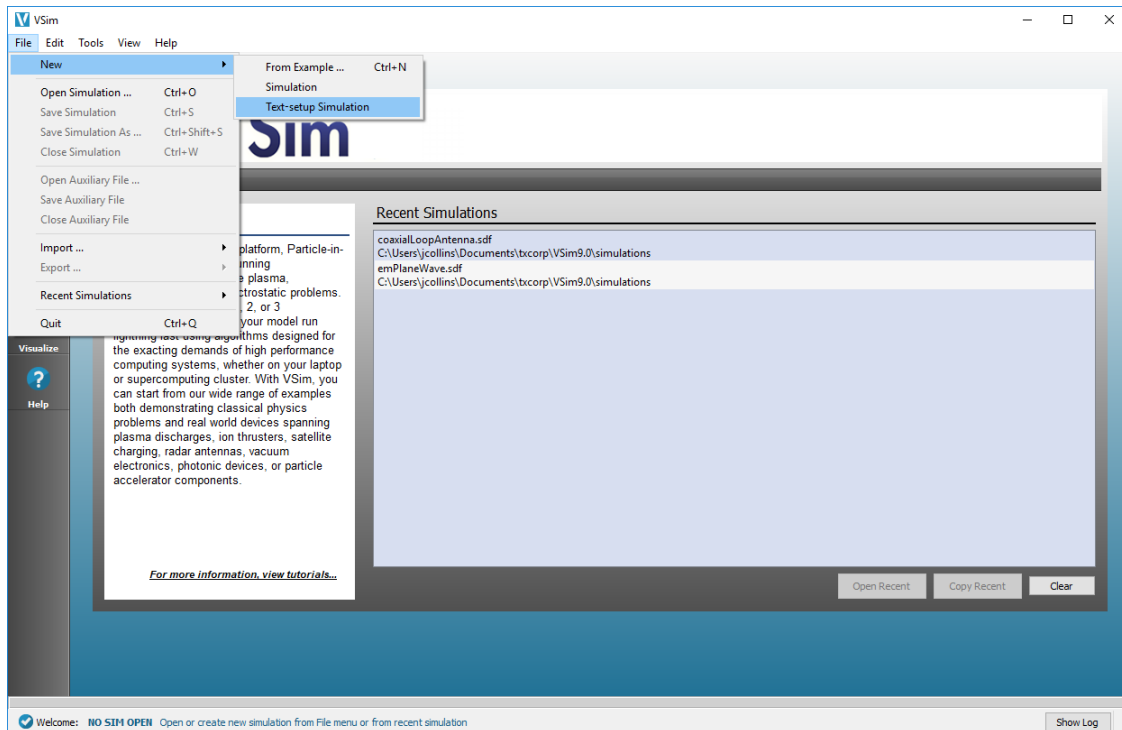
Fig. 3.9: Choose to create a new text setup simulation
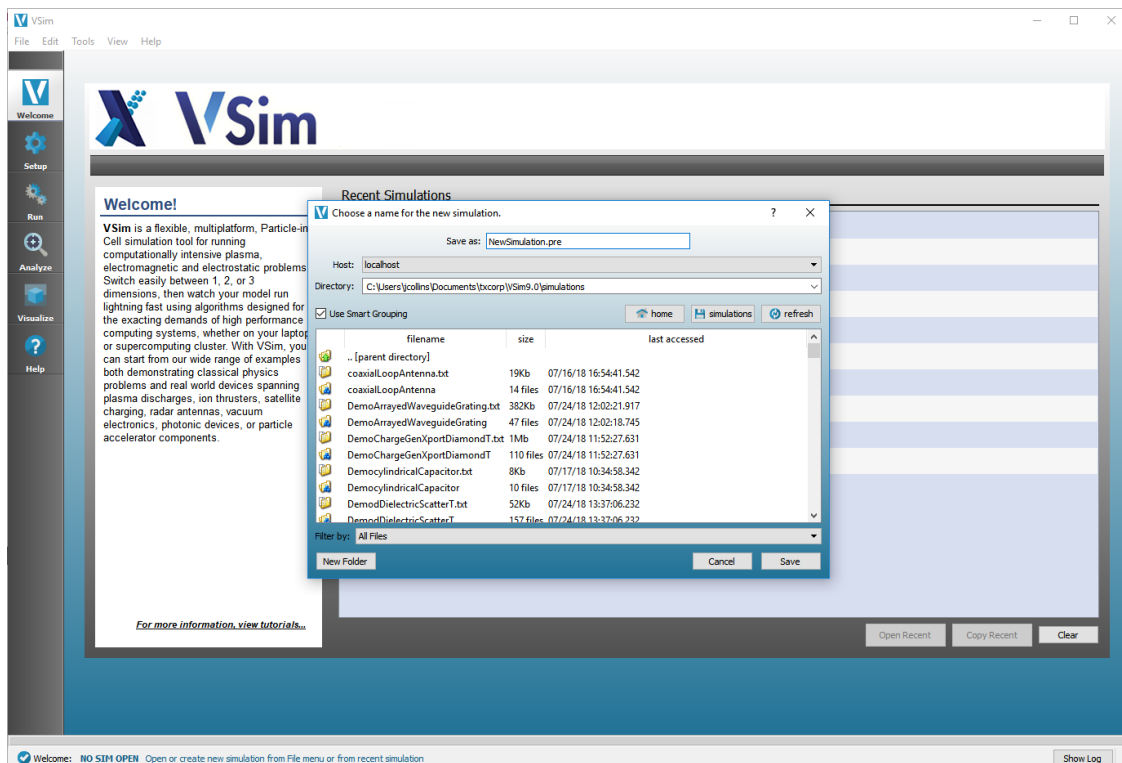
Fig. 3.10: Choose a name and directory for your new text simulation
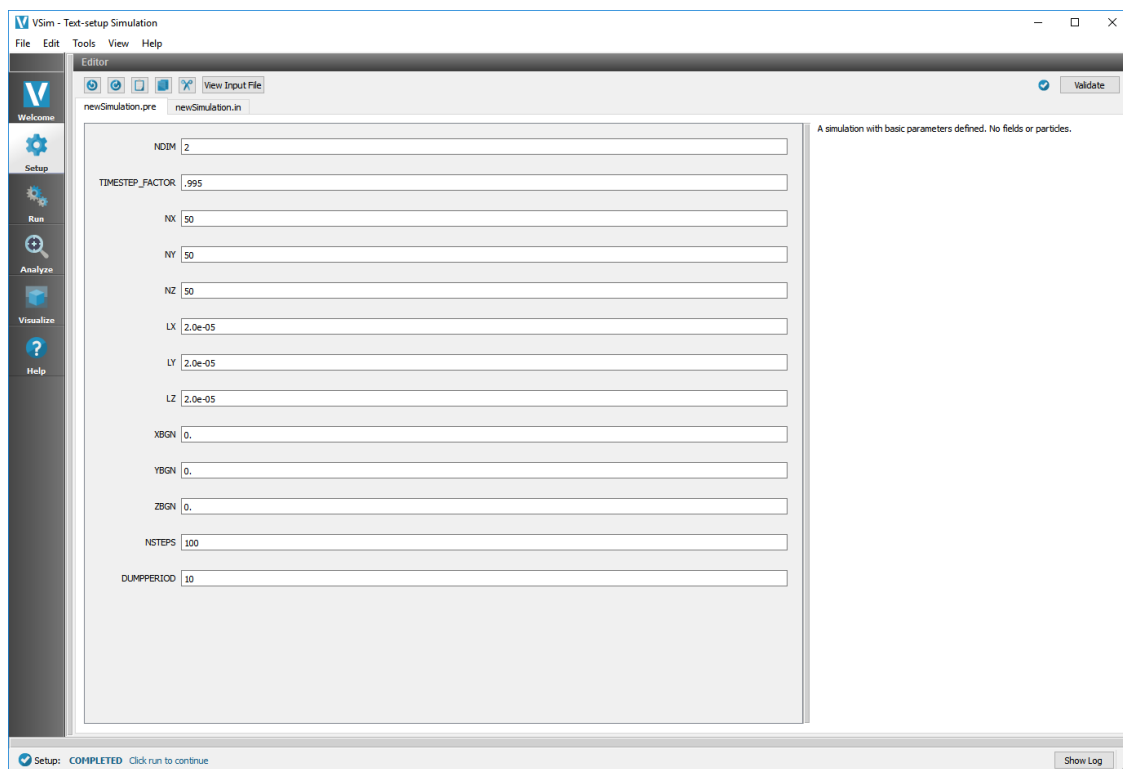
Fig. 3.11: A blank slate

MENUS AND MENU ITEMS

4.1 File Menu

The **File** menu contains options to control creating, opening, closing, and saving VSimComposer files and simulation directories. See Fig. 4.1.
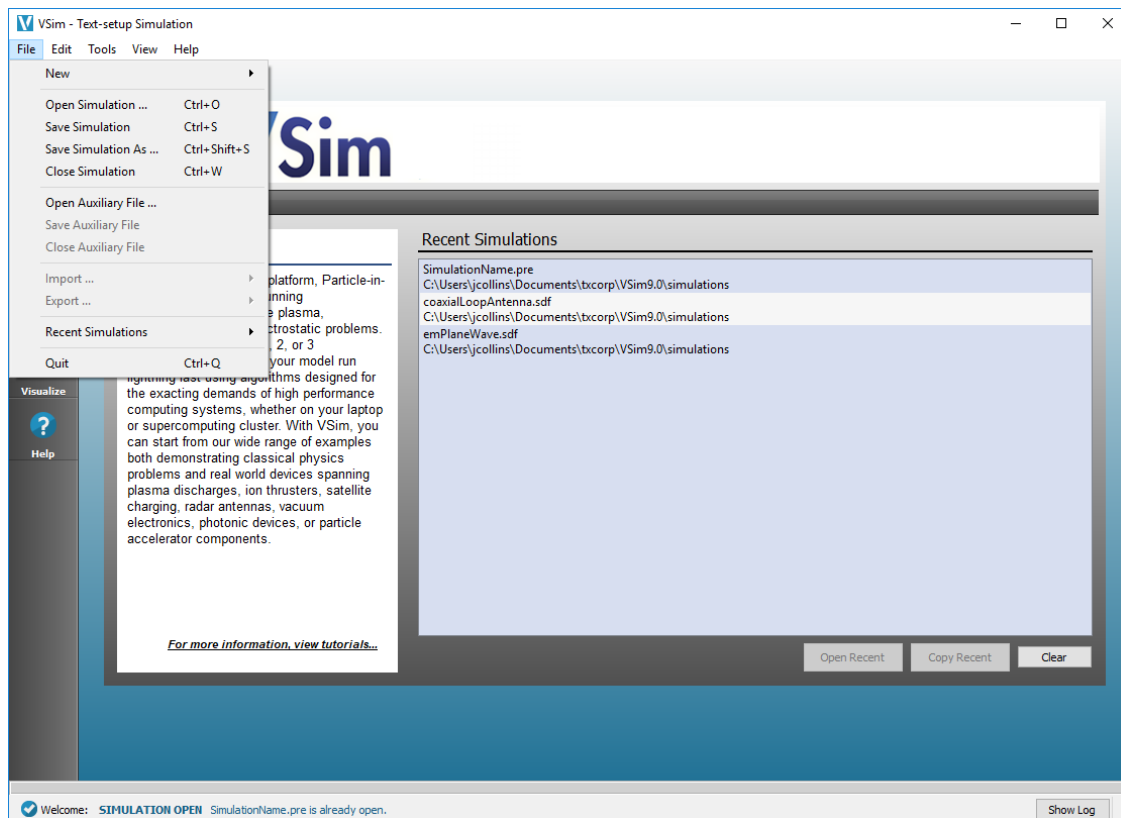
Fig. 4.1: File Menu

4.1.1 New

File -> New has three options:

- From Example

- Simulation

- Text-setup Simulation

For more information on each of these options, please see *Creating or Opening a Simulation*

4.1.2 Open Simulation

To open a directory where existing simulation files reside, select *Open Simulation* from the **File** menu. See Fig. 4.2.
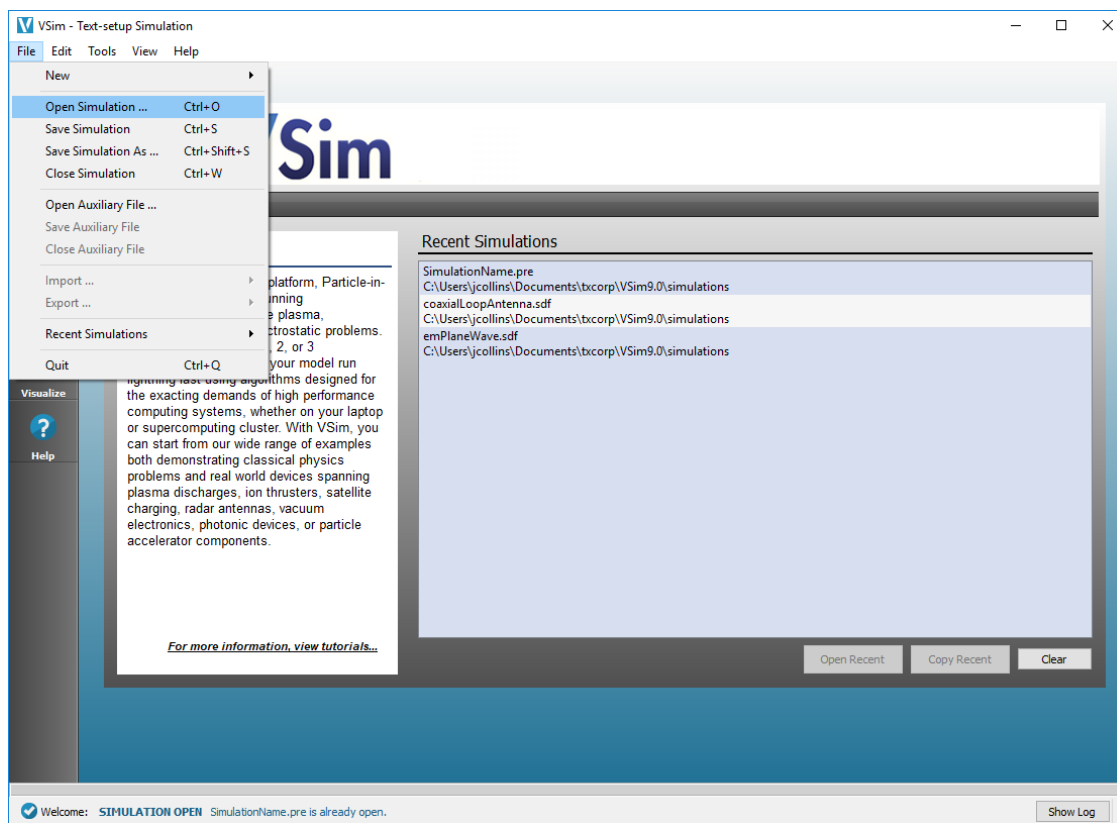
Fig. 4.2: Open selection from File menu

The default directory, **simulations**, is created for you during installation of VSim. You can modify this in the *Tools -> Settings -> Host settings -> Paths* menu. See Fig. 4.3.

4.1.3 Save Simulation

After a simulation is open, going to *File -> Save Simulation* allows you to save any modifications to the input file. The saving process will also run the Python pre-processor on the file in order to do any Python substitutions or calculations and to create the .in file.

4.1.4 Save Simulation As

After a simulation is open, clicking on *File -> Save Simulation As* allows you to save any modifications to the input file and write it in to a new file. The saving process will also run the Python pre-processor on the file in order to do any Python substitutions or calculations and to create the .in file.
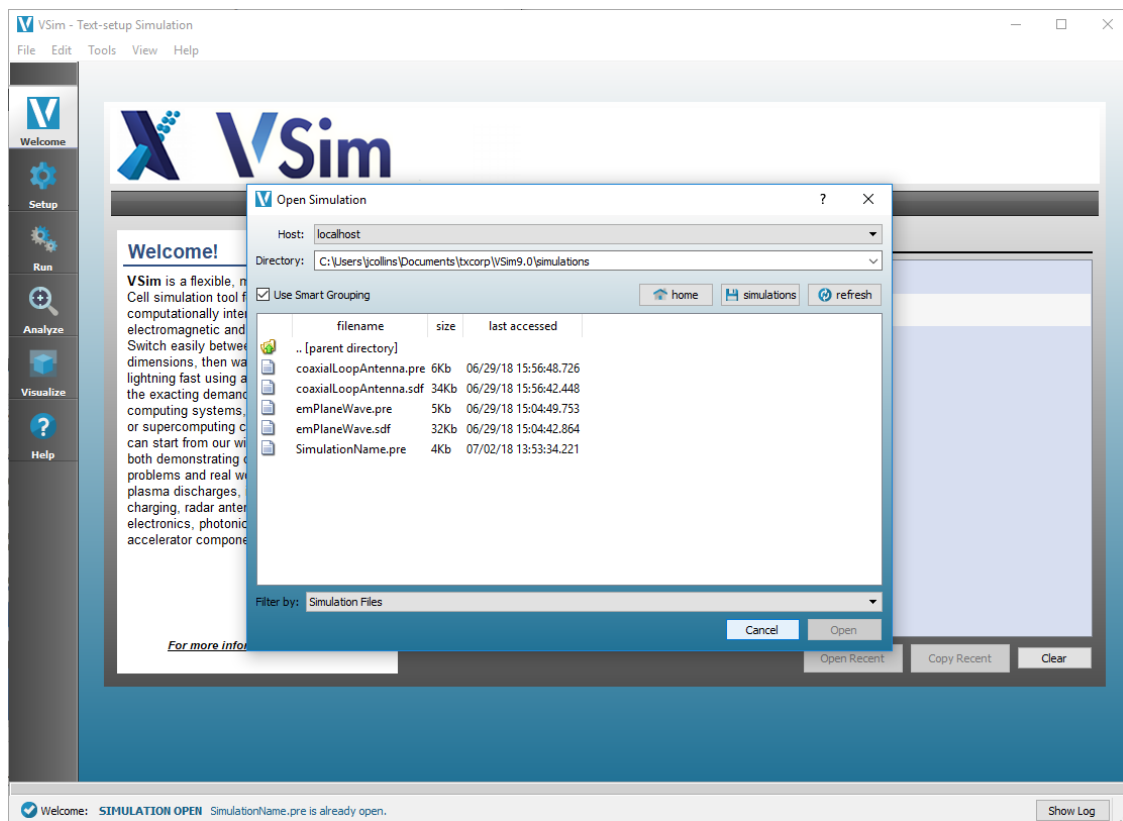
Fig. 4.3: Open Simulation window

4.1.5 Close Simulation

After a simulation is open, *File -> Close Simulation* allows you to close the current simulation and will return you to the **Welcome** window.

4.1.6 Open Auxiliary File

This can be used to open any file related to a simulation. It is most useful for opening and editing Python files. Many times, a simulation will have an associated Python file that contains script for post-processing in the **Analyze** window or for importing an external magnetic field.

4.1.7 Save Auxiliary File

This can be used to save any changes made to an auxiliary file.

4.1.8 Close Auxiliary File

This can be used to close an associated auxiliary file that was previously opened.

4.1.9 Import

When doing a visual-setup simulation, use the *File -> Import* option to import materials or geometries to use in your simulation.

Materials files must be of the extension ".vmat"

Geometry files can be any of .stl, .ply, .vtk, .stp, .step or .p12.

4.1.10 Recent Simulations

To access data to visualize from a recently conducted simulation, select *Recent Simulations* from the **File** menu in the menu bar. See Fig. 4.4.

Click on the name of the simulation whose data you want to open.

4.1.11 Quit

File -> Quit will close VSimComposer.

4.2 Edit Menu

The **Edit** menu contains commands that pertain to editing activities in the *Editor* pane of the VSimComposer window during **Setup**.

These operations are most useful when editing the input file directly. See Fig. 4.5.
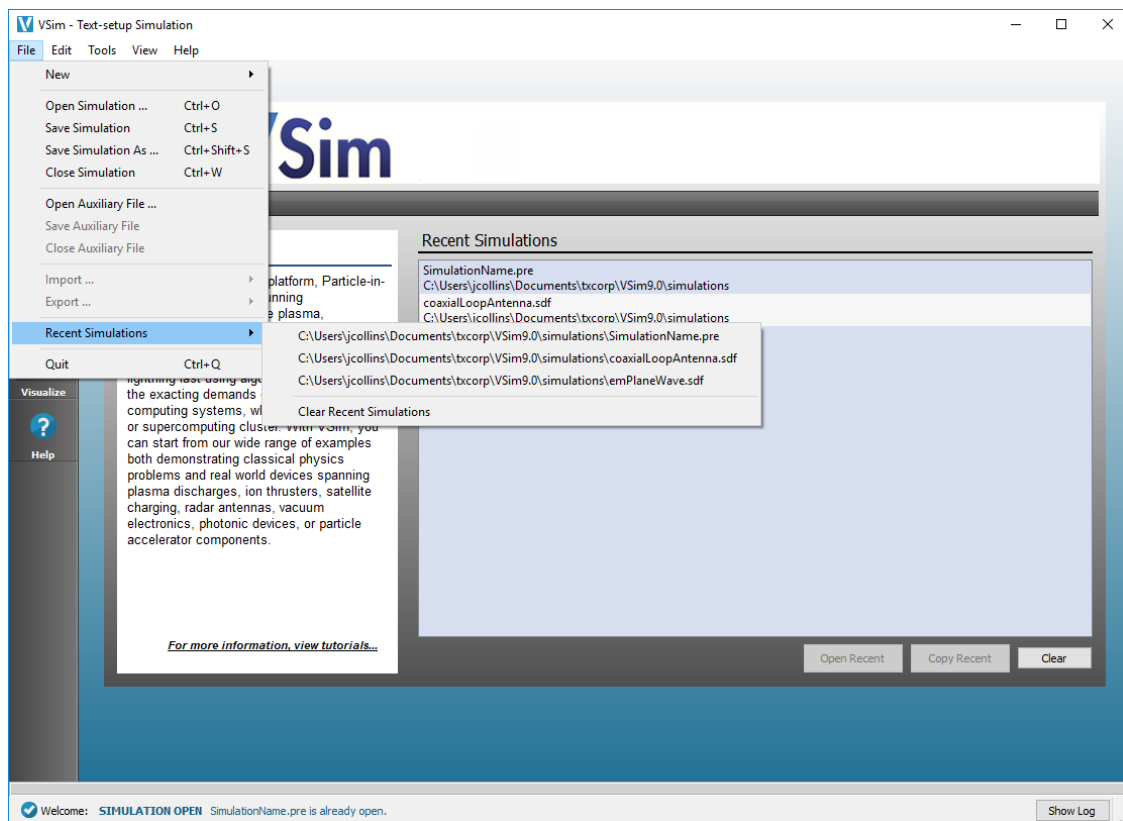
Fig. 4.4: Recent Simulation Selection in File Menu
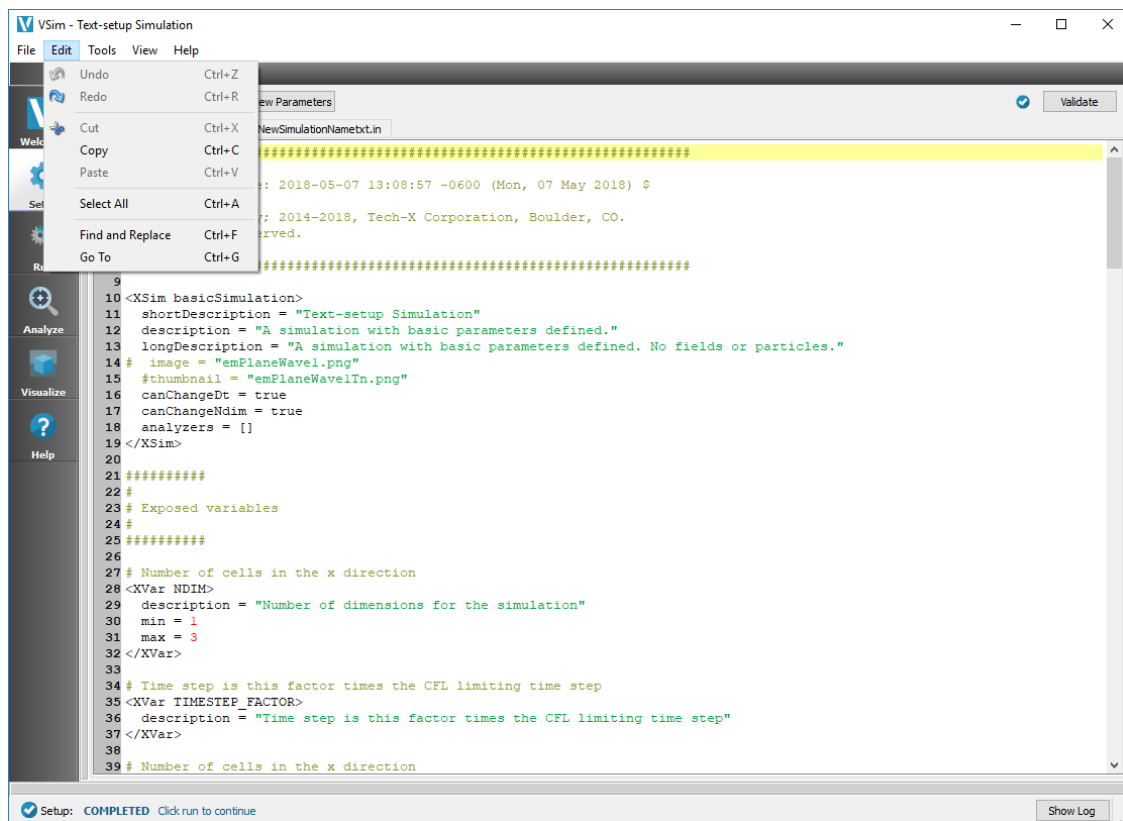
Fig. 4.5: Select Edit menu

4.3 View Menu

The **View** menu has the option for viewing the log file for VSimComposer. See Fig. 4.6.
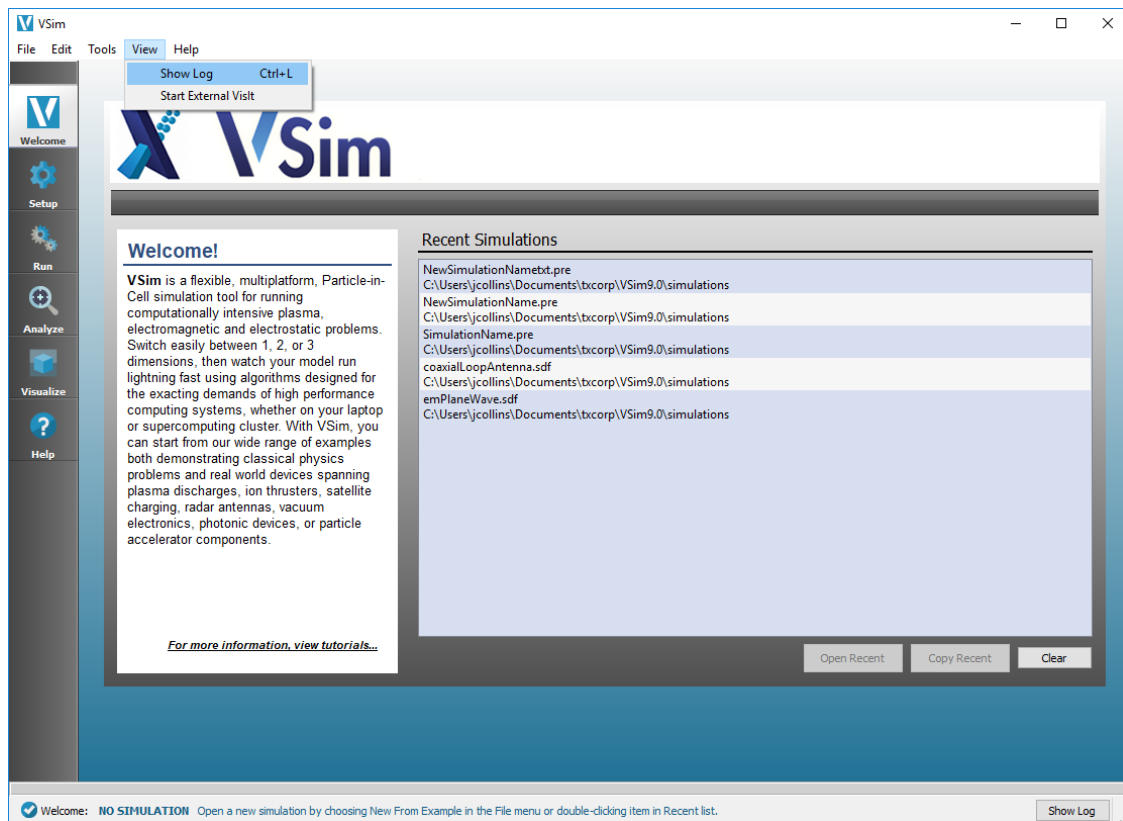
Fig. 4.6: View Log

This is useful if you run into problems and need help from the Tech-X Support team. It is a good idea to *Save to file* and send a copy of your log along with your message if you ever need assistance. See Fig. 4.7.

4.4 Help Menu

The **Help** menu contains both the VSim Documentation labeled *Help Contents*. It also includes pop-up information *About VSim*. See Fig. 4.8.

Selecting *Help Contents* is the same as selecting the **Help** icon in the left hand icon panel, and it opens the VSim Documentation Window in the users browser.

Selecting *About VSim* will bring up a pop-up menu that tells you information about the particular build of VSimComposer you have installed on your machine. It is very helpful to provide this information to Tech-X support personnel if you encounter any difficulties running VSim.

4.5 Tools/VSimComposer Menu (Settings/Preferences)

On Windows / Linux
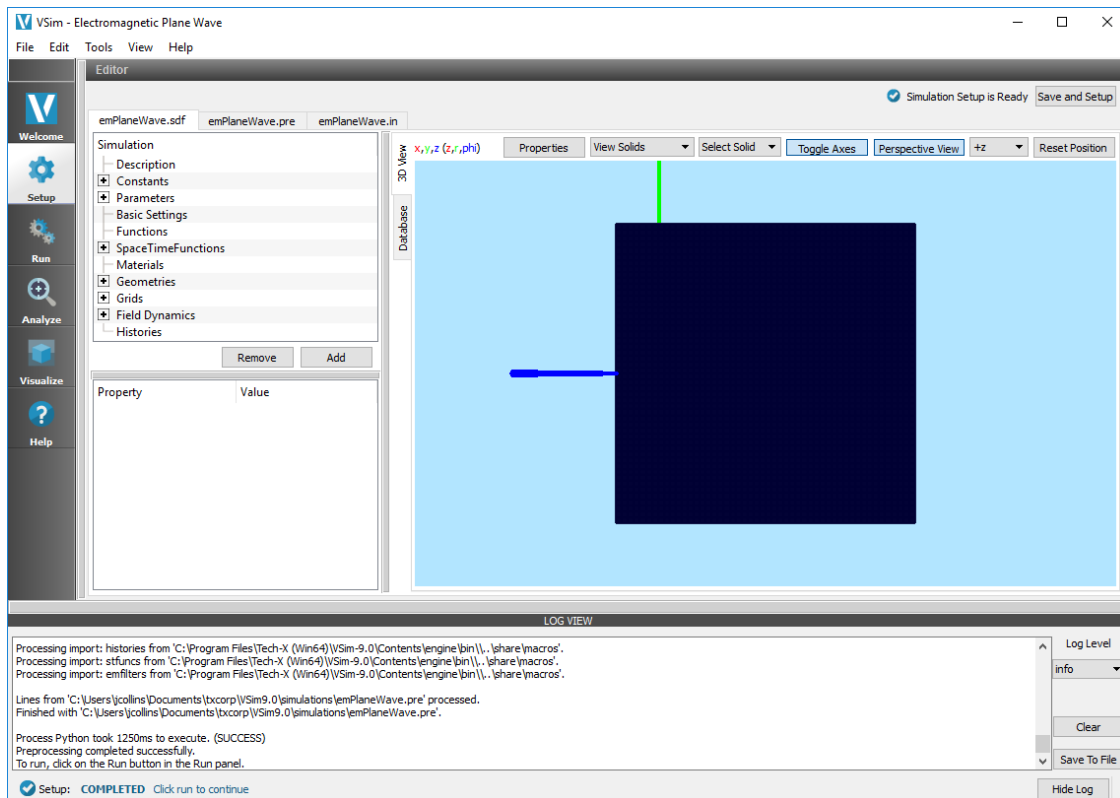
Fig. 4.7: Viewing the log contents from the view menu
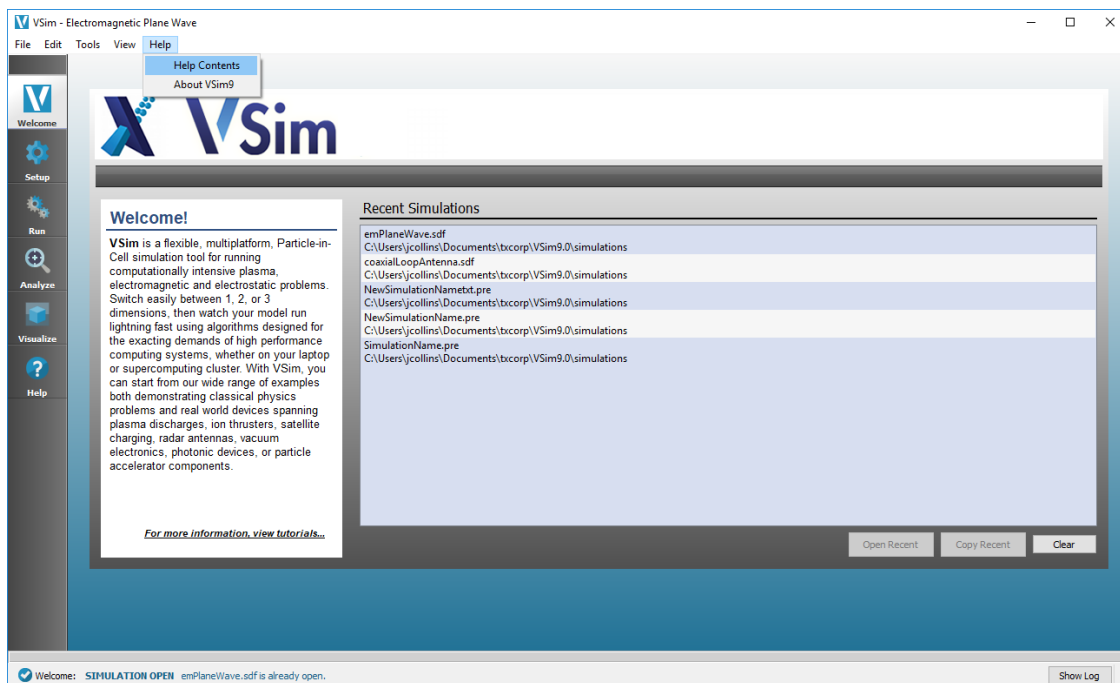
Fig. 4.8: Accessing help through the help menu

The **Tools** menu provides access to global settings for VSimComposer. The **Tools** menu contains the
Settings selection.

Select *Settings* from the **Tools** menu to access the *Application Settings* window. See Fig. 4.9.
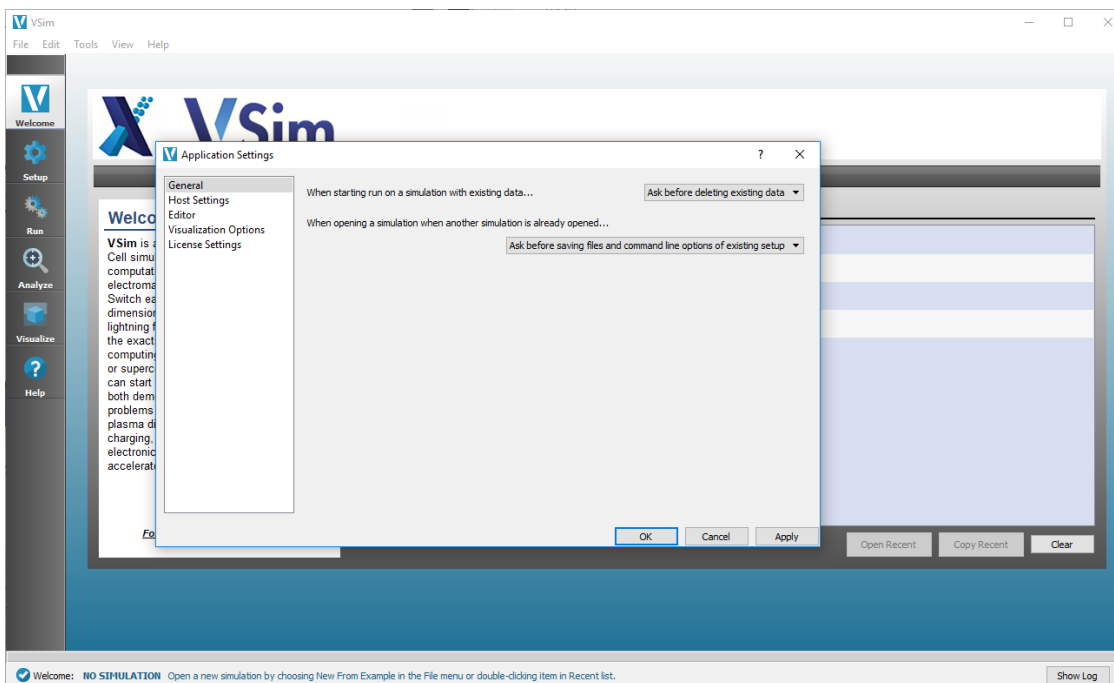
Fig. 4.9: Select Settings from the Tools menu

On Mac

The *VSimComposer* menu provides access to global settings for VSimComposer. The *VSimComposer*
menu contains the *Preferences* selection.

Select *Preferences* from the **VSimComposer** menu to access the *Application Settings* window. See Fig.
4.10.

4.5.1 General

General application settings set the default behavior for the VSimComposer file, directory, and other actions. See Fig.
4.11.

- When starting run on a simulation with existing data, the default setting that VSimComposer will use when
  starting a simulation that already contains data is *Ask before deleting existing data*. If you know that you will
  always want to create fresh data for each run, use the pulldown menu to set the default to *Always delete existing
  data*. If you know that you will always want to run on the data already available, use the pulldown menu to set
  the default to *Never delete existing data*. See Fig. 4.12.

- When opening a simulation when another simulation is already opened, the default setting that VSimComposer
  will use is *Ask before saving files and command line options of existing setup*. If you know that you will always
  want to save the simulation, this can be switched to *Always save files and command line options of existing
  setup*. If you know that you will never want to save the simulation, this can be switched to *Never save files and
  command line options of existing setup*.
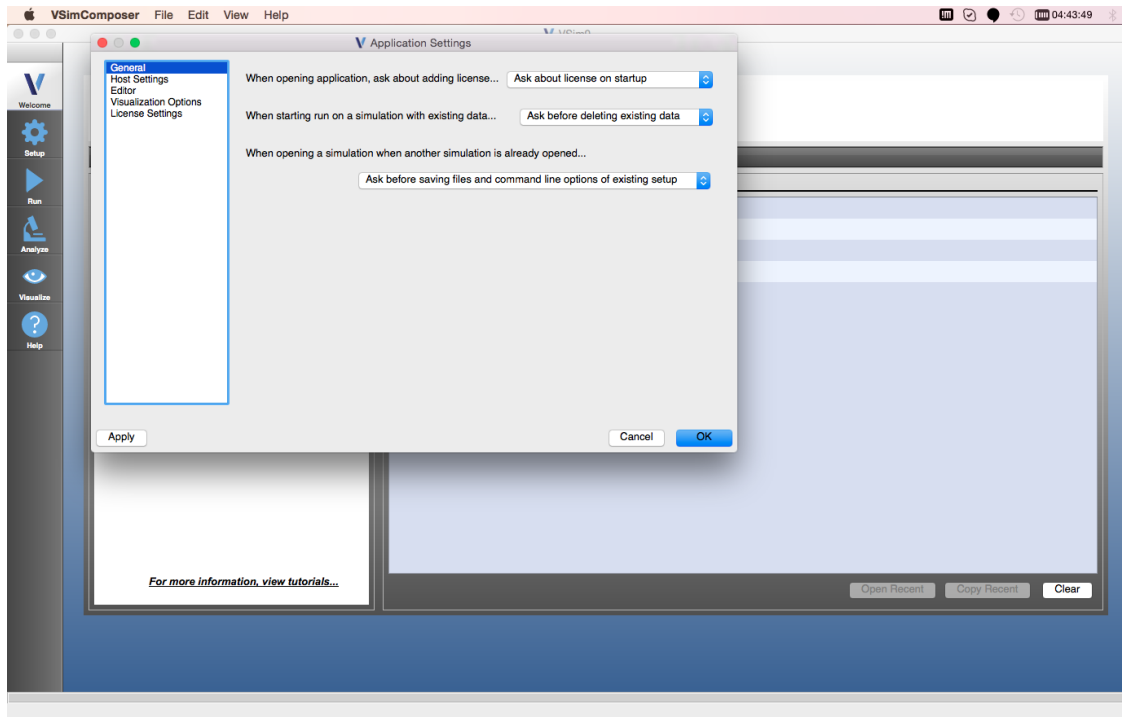
Fig. 4.10: Select Preferences from the VSimComposer menu
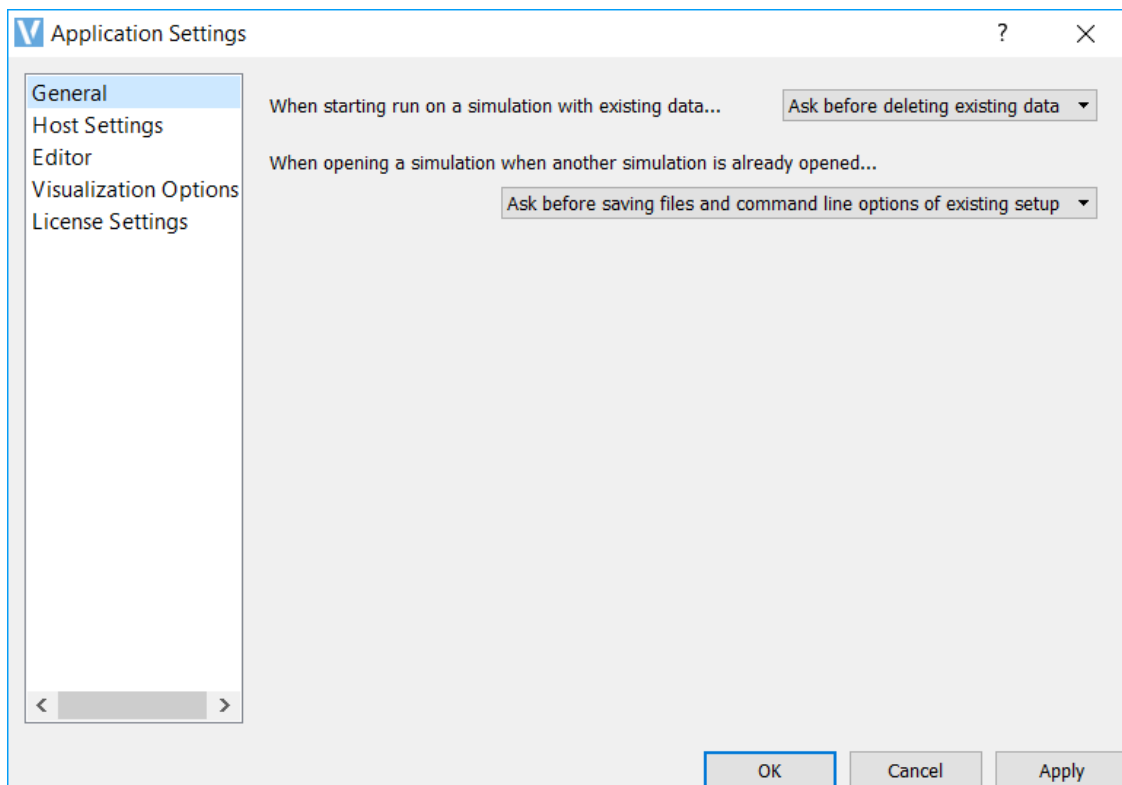
Fig. 4.11: General Application Settings
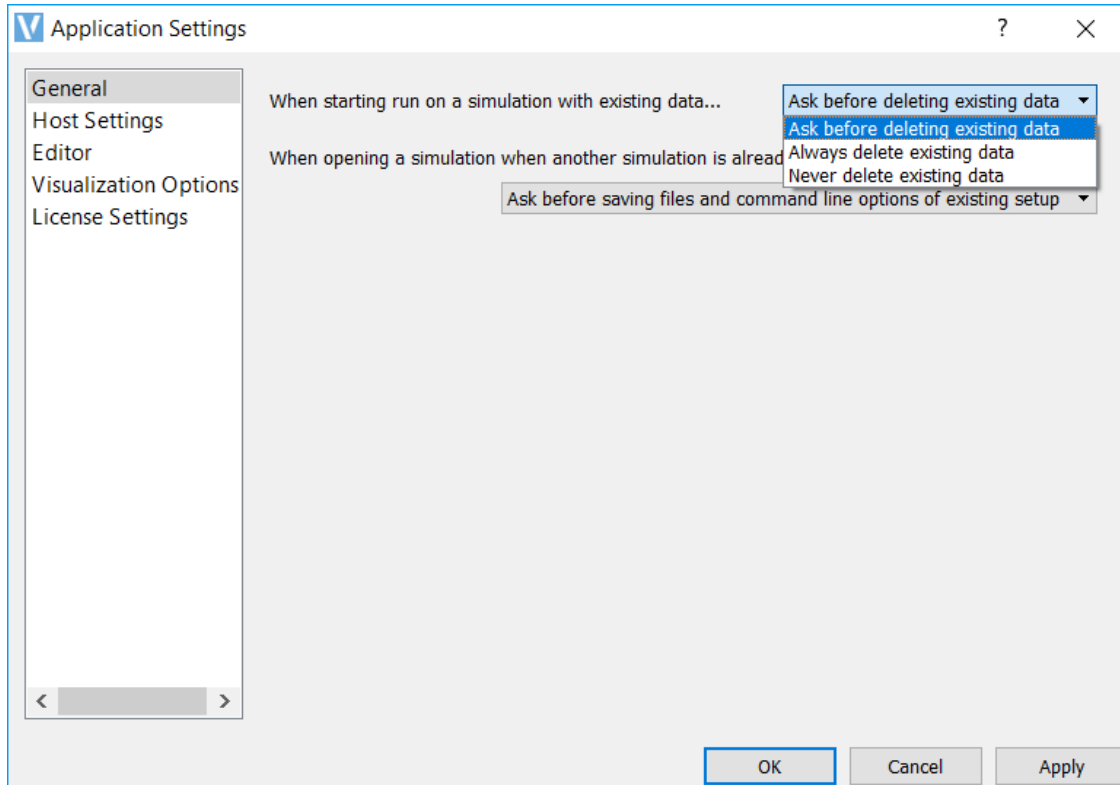
Fig. 4.12: Application Setting When Starting Run on a Simulation with Existing Data

4.5.2 Host Settings

The *Host Settings* section allows you to specify what machine to run on, the paths to your installation directory and workspace directory, and your preferences for serial or parallel simulations.

By default, you will be running on your localhost machine with the default installation directory and a preferred run method of serial.

General

Currently, VSim only allows for running simulations on the localhost. See Fig. 4.13.

Paths

- **Simulations directory** is the default directory for your runs.

- **Macros directory** houses the macros to be used in your runs.

- **Analyzers directory** houses the analyzers to be used in your runs.

- **Temporary directory** In some cases is necessary to set the environment variable TMPDIR (or TEMP on Windows) when executing vorpal. For example, on MAC when running in parallel and one gets an error "bind() failed on error Address already in use", then setting the TMPDIR to a shorter string can solve the issue.

See Fig. 4.14.
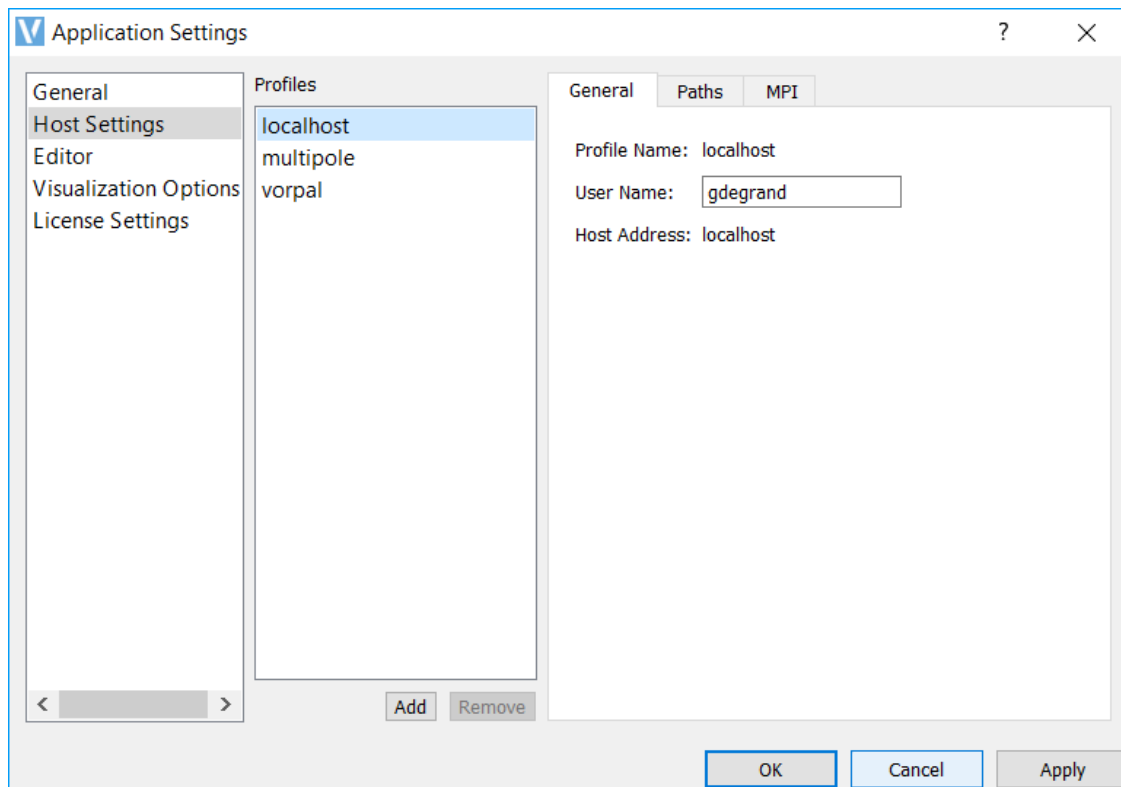
Fig. 4.13: Application Settings Host General Control Menu
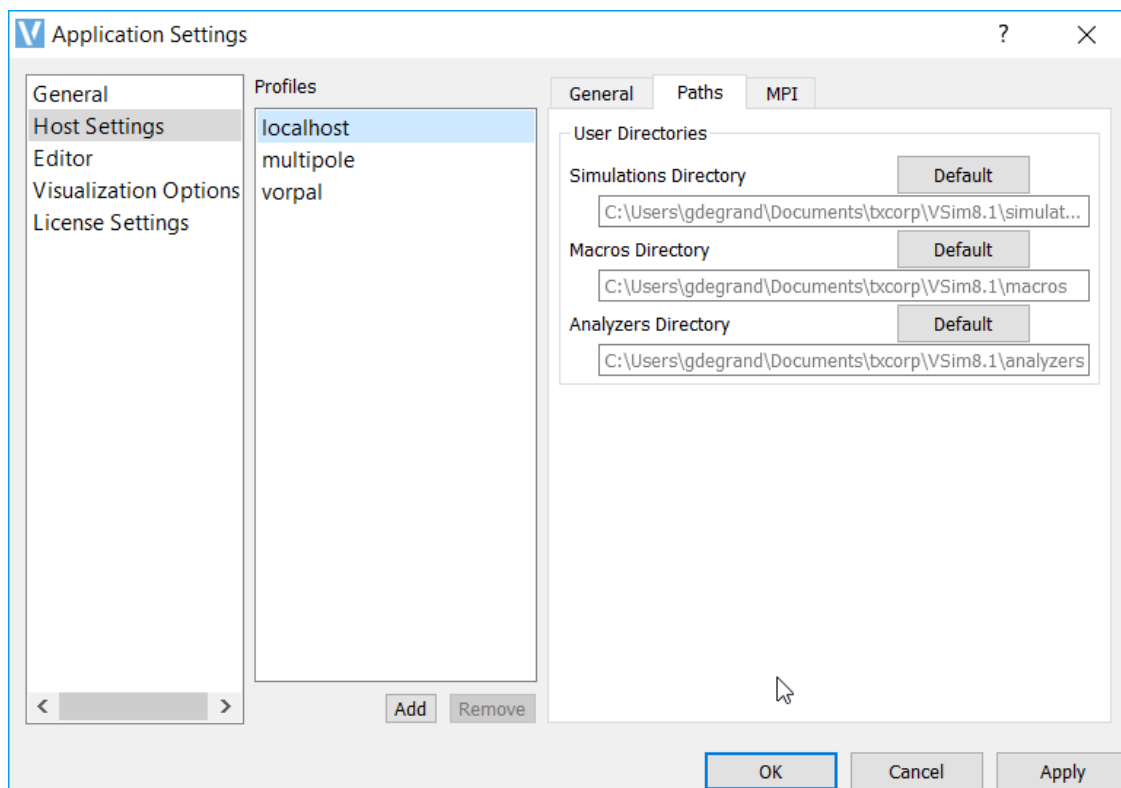
Fig. 4.14: Application Settings Host Paths Menu

MPI

- Preferred run method is the VSim serial engine (vorpalser) by default. If you have a multi-core system capable of parallel processing and a license activation file that is good on multiple cores, you can set the default to parallel instead of serial by clicking on the *Preferred Run Method* drop down menu and selecting parallel.

- Cores on machine shows the number of available cores for the current system that VSimComposer detects.

- **Preferred number of cores** is the field in which you may enter a new value and change the number of cores that will run simulations. This is helpful when you would like to run simulations using fewer processors than the number of cores for which your software is licensed, or perhaps want to try load balancing using more processes than you have cores. When the value in the *Preferred Number of Cores* field is set to something other than the last saved value, VSimComposer places an asterisk in front of the field label so that you are aware that you have changed the value and may wish to save the new value.

- Host File is where you can specify a file that contains the host nodes that you want to run on. This is useful if you have a large number of nodes, but need to run on a specific subset of them. For a description of how to create a hostfile see *Running Vorpal with mpiexec Using a Hostfile*.

See Fig. 4.15.
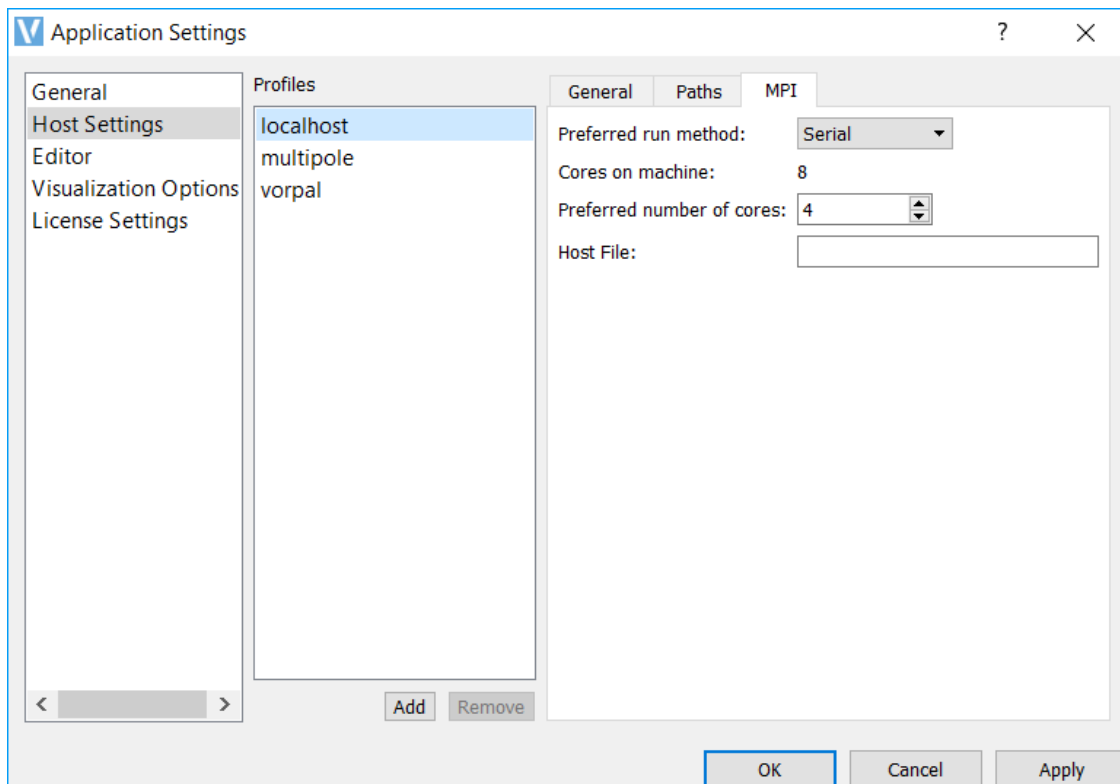
Fig. 4.15: Application Settings Host MPI Control Menu

4.5.3 Editor

The editor tab contains default settings for font size and a few other *Setup* tab options. These are editable to the users desired settings.

- **Files with Fixed-width Font**

> – Extensions refers to the file extensions that will obtain the following font and text size.

> – Font is where you can select the desired font.

> – Size is where you choose the desired text size.

- **All Other Files**

> – Font is where you select the desired font.

> – Size is where you choose the desired text size.

- Tabstop Width is the number of spaces that are inserted when the tab key is pressed.

- Color Style is where you select one of the choices for the color style of the text editor, including a white background, medium colored background, and dark background, with varying font colors.

- Open files in "Parameter" editor by default opens the file showing the editable parameters and an image overview (if this box is checked). When unchecked, the file is opened in the full text input file view.

- Use syntax highlighting, when checked, adds color to the text in the full input file view to help denote certain parts of the file. When unchecked, the text is all black.

- Show line numbers, if checked, shows the line numbers on the full text input file view. When unchecked, they are not.

- Highlight current line highlights the line where your cursor lies. If this box is unchecked, the line is not highlighted.

- Show post-processed file shows the post-processed .in file in a separate tab. This file shows the full text input file after it has gone through any Python calculations. When this option is unchecked, this file is not visible.

- Word wrap makes the text of the input file wrap at the end of the line. When this box is unchecked, the text will not be wrapped.

See Fig. 4.16.

4.5.4 Visualization Options

The visualization options tab allows the user control over default settings of the **Visualize** window in VSimComposer.

- Manual font sizing allows you to control the size of the fonts of plots.

- Enable VisIt context menu enables you to right-click on a visualization and open VisIt itself, where the user can access every function and feature of VisIt. It also enables the embedded point and line tools in VisIt as well as some of the generic view controls.

- Try harder to load cycles and times determines how aggressive VisIt is when opening dataset. When this option is checked (ON), VisIt will open every single dataset looking for time and cycle information. When this option is unchecked (OFF), VisIt will only look at the first file in a series. The advantage of having this option OFF is that datasets with lots of files are opened more quickly and with less memory usage. The disadvantage of having this option OFF is that the dump slider will not display any time or cycle information– only the dump number.

- Default ColorTable is the default color table used for plotting color plots.

See Fig. 4.17.

For more information on VisIt, please see: https://wci.llnl.gov/codes/visit/ and http://www.visitusers.org/index.php?title=VisIt_Wiki.
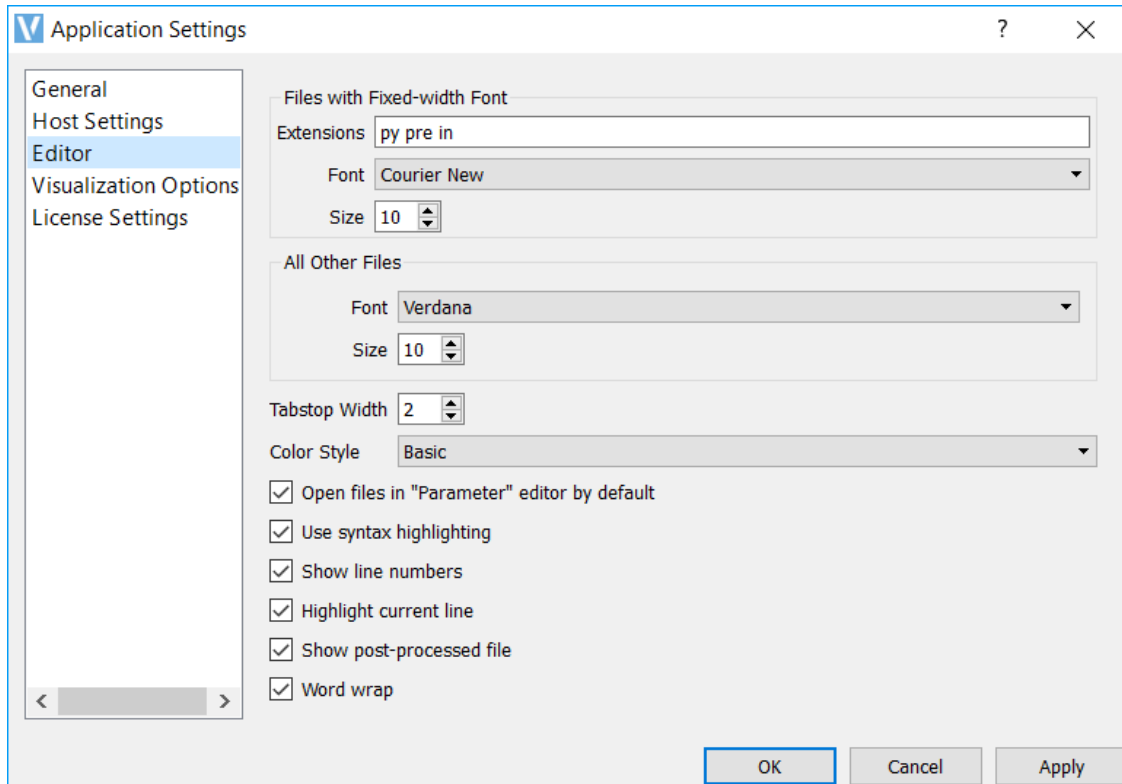
Fig. 4.16: Editor Menu
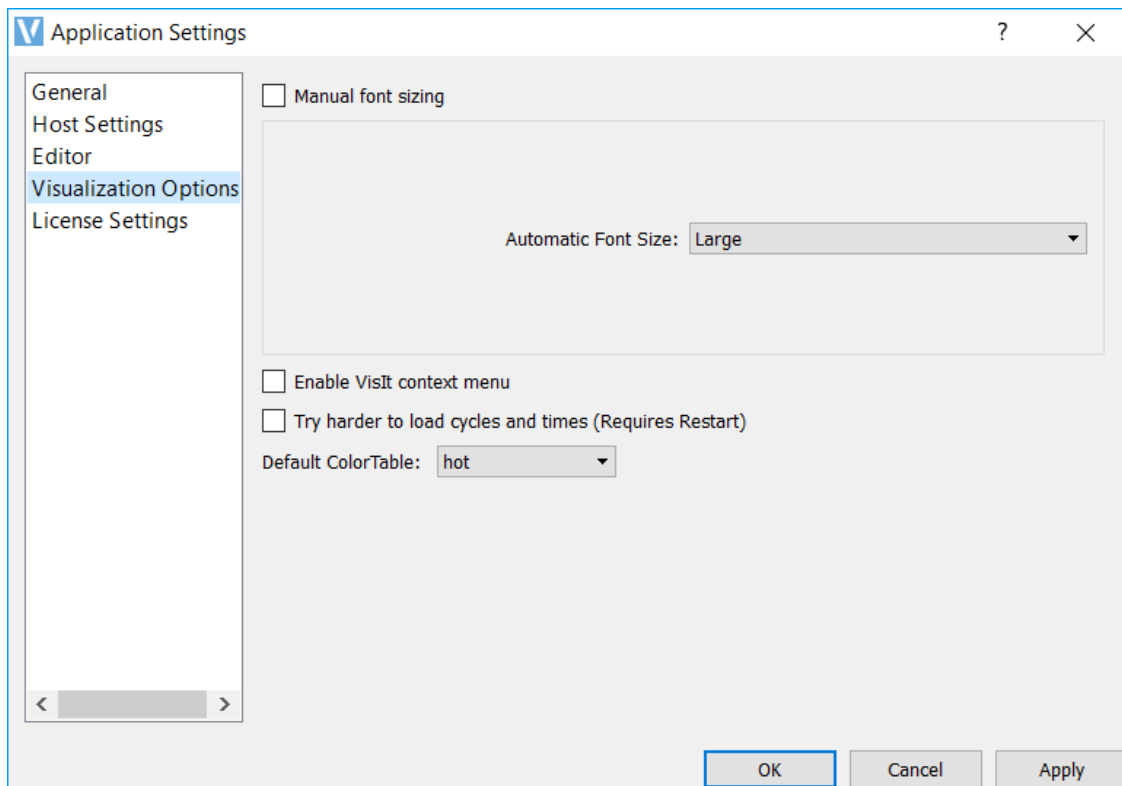
Fig. 4.17: Visualization Menu

4.5.5 License Settings

It is possible to review your license activation file and install a new license activation file if an upgrade or additional packages are purchased. To see the contents of the license activation file, click on the *Details* button. To install a new license activation file, click on the *Add* button. In the resulting file window, navigate to the previously-saved license file and then click the *Open* button. At this point, VSimComposer should import the license activation file and it will appear as the active license in the list of license files. See Fig. 4.18.
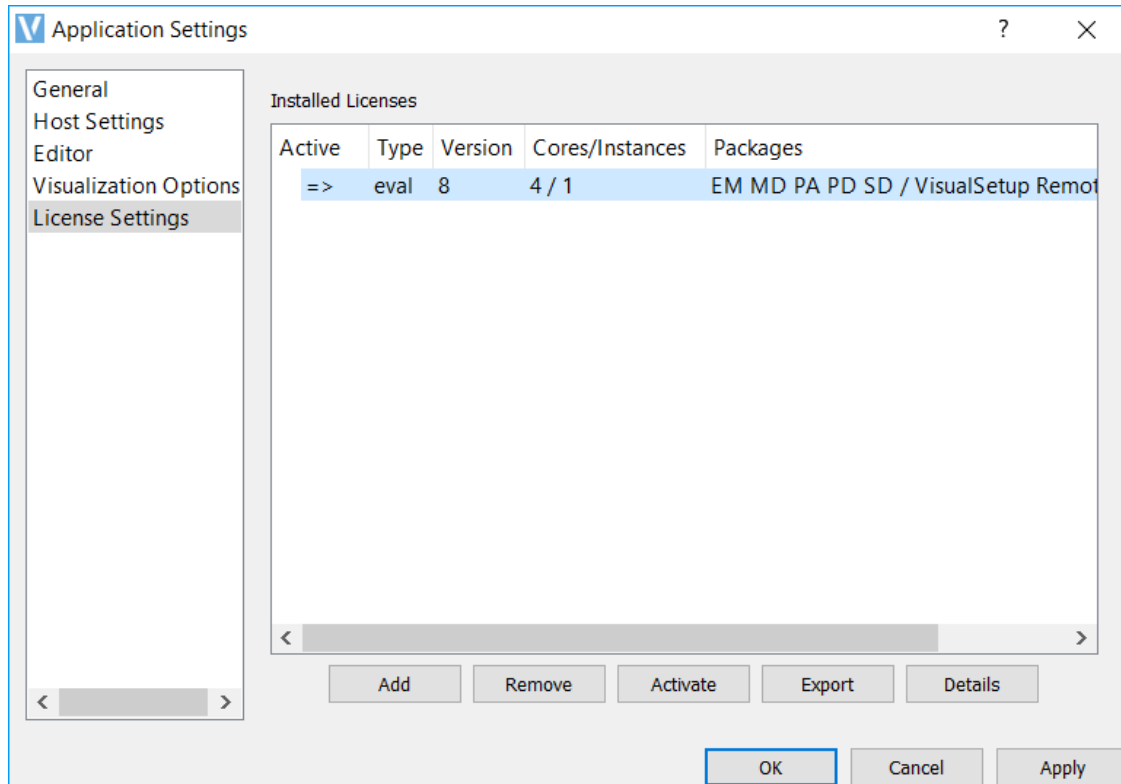
Fig. 4.18: License Activation File Menu

SIMULATION CONCEPTS

5.1 Simulation Concepts Introduction

VSim allows one to compute the dynamics of a system that has electromagnetic fields, particles, and material shapes that are advanced dynamically a *time step* at a time. A VSim simulation can contain some or all of the objects, with them interacting in various ways. In addition, the particles can be represented by *macroparticles*, which clump physical particles together so that one need not follow every individual physical particle, or by fluid fields.

The fields are defined on a structured grid in either cartesian or cylindrical coordinates. One can study just field dynamics, e.g., the propagation of electromagnetic fields on a grid, or solving for electrostatic fields for given boundary conditions and charge density, or fluid dynamics.

Material shapes, *geometries*, modify the dynamics of fields. For example, a conducting shape introduces an irregular region where the electric field vanishes. Consequently electromagnetic fields will scatter off of such a shape, and in electrostatics, such a shape will become an equipotential. Dielectrics shapes will modify the electric and magnetic.

Particles can be represented by *macroparticles* or a fluid. The particles interact with electromagnetic fields by interpolating the fields from the grid to the particle position. The particles then move for a given *time step* and deposit their contributions to the current and charge fields. This is the basic *Particle-In-cell* (*aka PIC*) algorithm.

Additionally, particles may interact with shapes. A shape can emit particles, absorb particles, or reflect particles. When one particle hits the surface, it may cause the emission of another particle, of the same or different kind. This can be secondary emission (the subsequent emission of an electron) or sputtering (the subsequent emission of a neutral atom).

Additionally, particles may interact with other particles through collisions. This is done through the Direct Simulation Monte Carlo (DSMC) method, which computes the effects of the collisions with each cell. Collisions may be elastic, where only momentum and energy are exchanged, or they may be inelastic, where kinetic energy is lost due to ionization or excitation of one of the particles. As well, particles may be created through field ionization, another sub-time-step process where the local strong electric field causes an atom to separate into an ion and an electron.

To bring the power of many CPUs to simulation, VSim makes use of distributed memory (MPI) parallelism. In this method, the simulation region is divided into domains (domain decomposition), with each process containing both its domain plus some grid cells beyond. The addition grid cells are known as *guard cells*. They are used in the communication between processes. Additionally they are used as the locations of particle sinks. Particles that leave a simulation must be removed or reflected back into the simulation to prevent crashes caused by out-of-range accesses of memory.

Simulation results are analyzed by looking at the generated data. In the regular course of a simulation, the simulation data is periodically dumped. As well, VSim allows the definition of *Histories*, which are time sequences of data. Examples include the Poynting flux through a surface or the number of particles absorbed by a shape.

In this section, we will begin by going over simulation concepts and properties, including:

- Grids

- – Decomposition and guard cells

- – Periodic boundary conditions

- Geometries

- Fields

  - – Electromagnetic fields

  - – Electrostatic fields

  - – Planar boundary conditions

  - – Conformal boundaries

- Particles

  - – Macroparticles

    - * Particle-in-cell simulation

    - * Particle sources

    - * Particle sinks

  - – Fluids

- Reactions

- Histories

5.2 Grids

The grids used by VSim are structured, coordinate aligned, where the grid lines are along coordinate directions. Such a two-dimensional grid is shown in Fig. 5.1. One can choose either a uniform spacing or a non-uniform spacing as shown in Fig. 5.1, and the coordinates may be either cartesian or cylindrical. In Fig. 5.1, each of the cells is numbered by its indices. In this $2D$ case, there are two indices; in general one for each direction. The cell indices start at 0 and end in the x direction at $NX - 1$ for a grid that has NX cells in the x direction. For a $3D$ grid, there would be another direction out of the page.

A cell of a 3D grid is shown with z coming out of the page in two views in Fig. 5.2. A cell owns its interior plus the interior of its lower face in each direction plus, the interior of its lower edge in each direction, plus the lower node of its owned edges. The owned node is circled in both views of Fig. 5.2. The owned edges are shown on the left side of Fig. 5.2, with the x-edge red, the y-edge green, and the z-edge blue. Similarly, the owned faces of a cell are shown on the right side of Fig. 5.2, with the x-normal face red, the y-normal face green, and the z-normal face blue.

In FDTD EM, the concept of a dual grid is useful. The dual grid is the grid with nodes at the centers of the regular grid. The edges of the dual grid pierce the faces of the regular grid and vice-versa.

5.2.1 Guard cells

Guard cells, cells just outside the simulation grid, are needed for having sufficient field values in the simulation region, for particle boundary conditions, and for parallel communication (to be dicussed later). An example of the first case is where a field must be know at each of the nodes of the simulation. Then, since the last node in any direction is owned by the cell one beyond the simulation, the cells one beyond the *physical grid* must be in the simulation. Thus, the grid must be extended by one cell in the last of each direction, as shown in Fig. 5.3.

Fig. 5.1: Structured 2D grid.
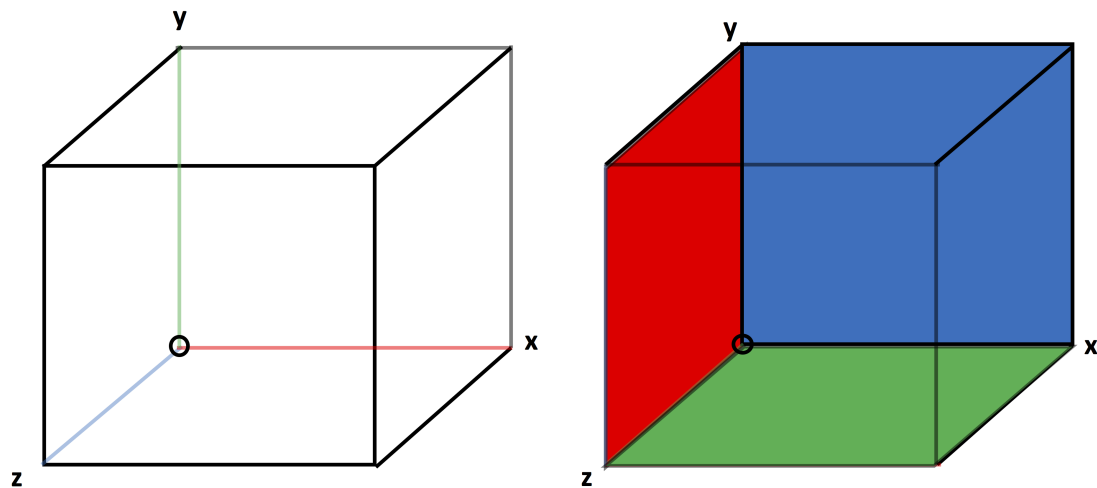
Fig. 5.2: 3D cell in a view showing its edges and a view showing its faces.

Fig. 5.3: Grid extended to include the upper nodes, which belong to the cells one past the last cell in each direction.

Note: The user-defined grid is called the physical domain. The grid extended by Vorpal is called the extended domain.

For particle boundary conditions, the grid must be further extended down by one cell in each direction. When a particle leaves the physical domain, it can end up in one of these additional cells, which can be either above or below. A data value associated with that cell determines what to do with the particle, e.g., absorb it (remove it), reflect it, or carry out some other process. The associated extended grid is shown in Fig. 5.4. The physical cells are depicted by the red grid. The associated dimensions are in blue. The extended cells (shown in green) enclose both the physical cells and the guard cells added by Vorpal.

5.2.2 Periodic Boundary Conditions

Periodic boundary conditions can be used to control both field and particle behavior at the edge of the simulation domain. In the case of particles, periodic boundaries ensure that particles leaving one side of the domain reappear at the opposite side. For example, particles traveling at a speed of $-v_\phi$ will go through $\phi = 0$ and reappear at $\phi = 2\pi$. Fields, on the other hand, will be copied from the plane at index 0 to the plane at NX and from the plane at $NX - 1$ to the plane at -1, i.e. from the last physical cell to the guard cell on the other side.

5.2.3 Parallelism and Decomposition

Parallel (distributed memory, MPI) computation is carried out by domain decomposition. A particular decomposition is shown in Fig. 5.5. In this case, this is a decomposition of a rectangular region by the red lines, with the individual subdomains each give a unique index. However, Vorpal can simulate any region that is a non-overlapping collection of rectangles (appropriately generalized for 3D and 1D), with each rectangle being a subdomain of the decomposition.
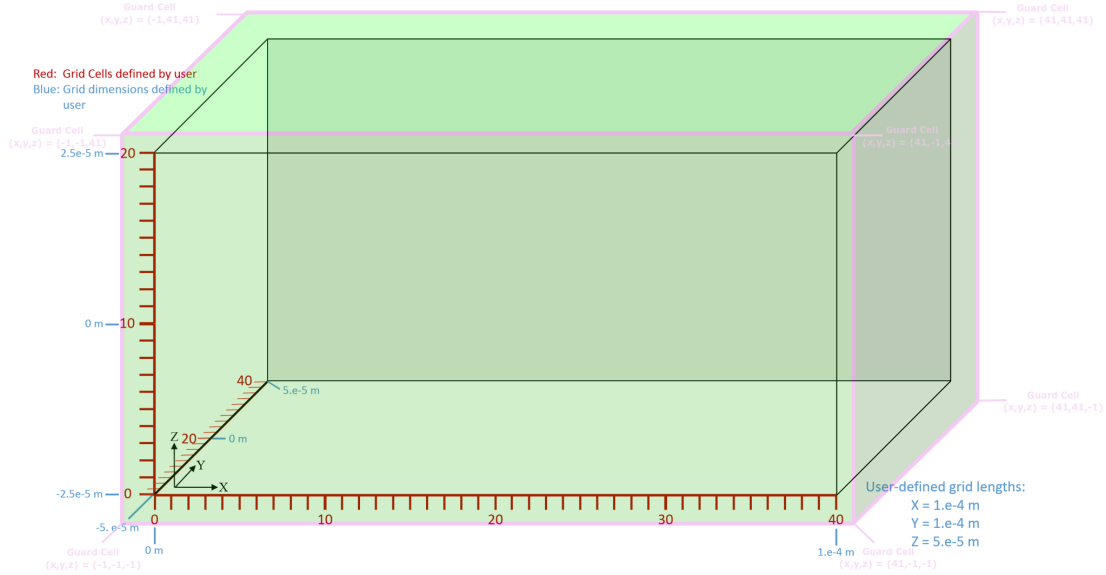
Fig. 5.4: Cartesian grid extended by Vorpal
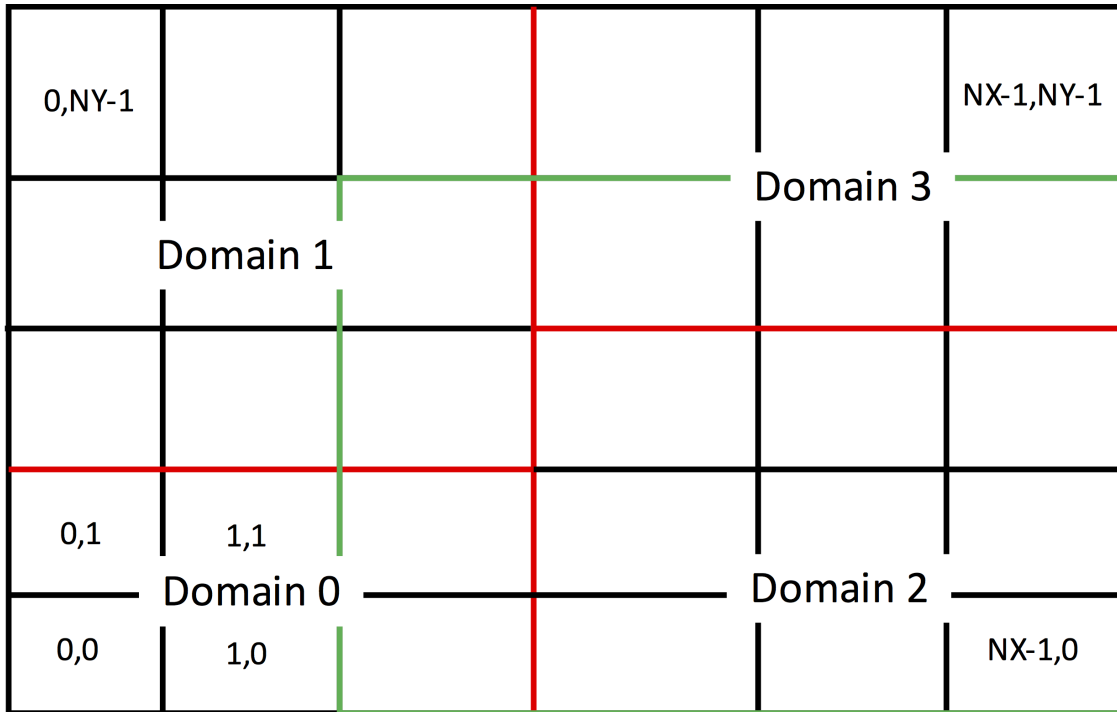
Fig. 5.5: Parallel decomposition of a computational domain.

For the most part, parallelism and decomposition are handled under the hood, but it is useful to understand a few concepts. In Fig. 5.5 one can see a green rectangle that extends one cell more into the simulation region beyond Domain 2. Fields in the cells of this overlap region are computed by the subdomain holding the cell, but they have to be communicated to Domain 2, as it needs this boundary region to update its fields on the next time step. On the other hand, particles may leave Domain 2 and end up in one of the cells still inside the green rectangle. Those particles must be sent to the processor holding the cell they are in for further computation.

For the above situation to work, the field update method for a given cell must not need information more than one cell away. The standard updates for electromagnetics and fluids indeed have this property. As well, the particles must not travel more than one cell in a time step. This is true for explicit electromagnetic PIC with relativistic particles, as the Courant condition prevents the time step from being larger than the time it takes for light to cross any cell dimension, and relativistic particles travel slower than the speed of light. Finally, the particles must not interpolate from fields more than one cell cell away, nor must they deposit current more than one cell away. This is true for the simplest interpolation and deposition methods.

However, if you have electrostatic particles that travel more than one cell per timestep, or particles that have a larger deposition footprint, then manual setting of some parameters may be necessary. In particular, the grid parameter, `maxCellXings`, states the maximum number of cells a particle might cross in a simulation, and the parameter, `maxIntDepHalfWidth`, provides the width of the deposition stencil. From these follow the `overlap` needed for the subdomains. These parameters are discussed in more detail later.

5.3 Geometries

Geometries in Vorpal are non-grid-aligned material shapes. They can be defined in a number of ways, e.g., a triangular surface mesh from an STL file, a set of shape from a STEP file, Constructive Solid Geometry (CSG), and functional (a function defining whether a point in space is inside or outside). In visual setup, the material assigned to the shape determines how the electromagnetic field interacts with the shape.

Visual Setup supports STL import with translation, STEP import, and CSG. One can then assign a material to the shape. CSG is discussed in the *Visual Setup* section, where you learn to build primitives and perform operations on them.

In text setup, you can still use CSG primitives, but the process is a little different. Rather than being able to simply click and add primitives, text-setup requires one to import the appropriate `geometries` macro and define primitives using built-in functions. This all will be covered in detail in the *Geometries* page of the following *Text-based (.pre) Input File Structure* section. Also covered in this section are the procedures for importing CAD and Python-defined geometries, moving and rotating shapes, and building custom primitives for your simulation.

One can also use geometries in particle boundary conditions. Particles can be emitted, reflected, or absorbed by a geometry. Additionally, an absorber on a geometry can be attached to a secondary emitter or a sputterer as a source for the same or another kind or particle.

5.4 Electric and Magnetic Fields

The electric and magnetic fields are the most important fields in VSim, as they impact the motion of charged particles. In this section we discuss how fields generally work, then we discuss the field update concept. We then discuss the particulars of the updates for each of electromagnetics and electrostatics.

5.4.1 Field Basics

The general concept of a field is a scalar, vector, or tensor that is a function of space and time. It is most commonly implemented in VSim by values on a grid, i.e, a value for each cell. For finite difference methods, the different

components (e.g., vector components) have a location, i.e., place where they most accurately represent the field value, within each cell. For a scalar field, like the electrostatic potential, the location is either at the node (lower corner) or the cell center. For a vector field, the natural locations are either at the centers of the edges or at the centers of the faces. Hence, a field in VSim has a property, `offset`, that determines this offset.

Fields can be messaged as needed and described in *Parallelism and Decomposition*. In VSim, fields are messaged according to their overlap, and they are always messaged both up and down.

Some fields are used for deposition of charge and/or current. These deposition fields come with a few changes. First, they are automatically zeroed at the beginning of each time step. Second, to get the charge and current correct in a parallel simulation, the contributions in the guard cells on one domain have to be sent to the owning domain and be added into the charge or current on that domain.

5.4.2 Field Updating Basics

Fields can be either static or dynamic. E.g., the magnetic field in an electrostatic simulation does not evolve. It is set once, and that field is used throughout the simulation. On the other hand, the electric field in an electromagnetic or electrostatic simulation changes at each time step, and the magnetic field in an electromagnetic simulation also changes at each time step. In addition, a field may be made up of a static part and a dynamic part, in which case the two are added together to get the total field at each time step.

The update of a field can be either explicit or implicit. Explicit means that one can write the new field at a given cell in terms of the old field values. Implicit means that one must solve an equation for the new field values. An example of the latter, to be discussed in more detail. In either case, one is updating the field, and the object that does this is known as an *updater*.

5.4.3 Electromagnetics

In electromagnetic simulations the electric and magnetic fields obey Maxwell's equations, i.e., Ampere,

$$\frac{d\vec{E}}{dt} = -\frac{1}{\epsilon_0}\vec{J} + \frac{1}{c^2}\nabla \times \vec{B}$$

and Faraday.

$$\frac{d\vec{B}}{dt} = -\nabla \times \vec{E}.$$

These are discretized and updated using the Yee algorithm [Yee66]. In that algorithm staggered grids are used, which is equivalent to saying that the electric field is an edge field, and the magnetic field is a face field, following *Grids*. The Yee algorithm can be thought of as using the integral formulation of Maxwell's equations on the faces of the grid (for B) and the faces of the dual grid (for E). The corresponding updaters are the `YeeAmpere` and the `YeeFaraday` updaters. The update for pure EM is staggered in time, as in leap-frog integration.

The update region is over the interior of the simulation. This varies for different components of the electric field and is illustrated in Fig. 5.6. The electric field in the x direction, shown in red, lies on x edges. The interior x edges have lower-left cell of (0,1) and upper-right cell of (NX-1,NY-1). In Vorpal, the region is greater than or equal to the lower bounds and less than (not equal to) the upper bounds. Hence, the update slab for E_x is [(0,1),(NX,NY)]. By similar reasoning, now referring to the green arrows, the update slab for E_y is [(1,0),(NX,NY)].

For the magnetic fields, there is a similar difference in the update region per component. The magnetic fields are face fields, and so they are updated on any face that is in the interior or on a boundary.

When dielectrics are present, Maxwell's equations need to be modified. Ampere's equation gives the new value of the displacement, D, from which one obtains the new value of the electric field, E by multiplication by an effective inverse dielectric tensor. Accurate algorithms for the time domain are described in [WBC13][WC07], while a more rigorous, but frequency domain algorithm is described in [BWC11].
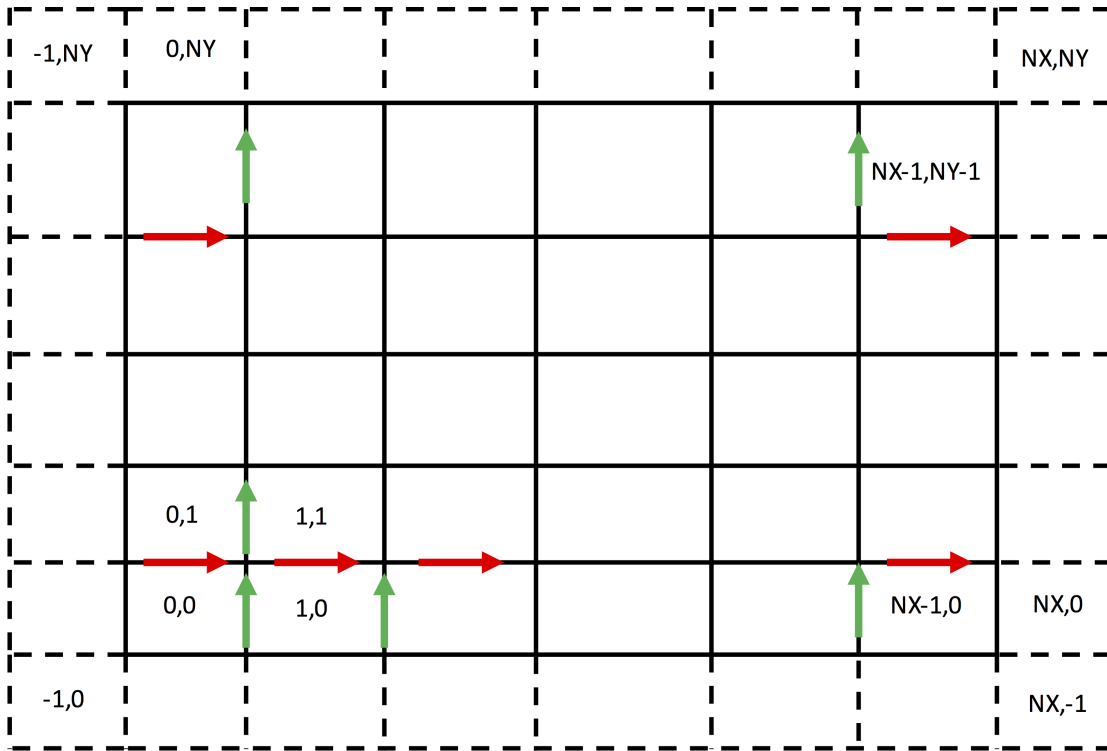
Fig. 5.6: Update region for the electric field.

Electromagnetic Slab Boundary Conditions

Beyond the edges of the simulation, values of the electric field are not known. Hence, one cannot update the electric field at the edge, as that would require a difference with an unknown value. Instead one sets boundary conditions. For the simple case of a purely rectangular region, the boundaries that must be set are shown in Fig. 5.7.

They are simply the tangential values of the electric field on the boundary. Just as in continuum EM, one need not set boundary conditions on the magnetic field. Setting the values of the tangent electric field at boundaries is sufficient.

Electromagnetic Conformal Boundary Conditions

Electromagnetic boundary conditions are to some degree setting the value of the electric field, but they can also involve a modification of how the magnetic field is updated.

In Fig. 5.8 is shown a curve the interior (lower and left) of which is vacuum, while outside is Perfect Electric Conductor (PEC). All electric fields on edges that are totally in the PEC are set to zero. Then there are two approximations. In the *stair-step* approximation, one additionally sets to zero all electric fields whose edges are more than half outside. In the *Dey-Mittra* [DM97] algorithm, one starts by keeping any electric field on an edge even partially outside of the PEC. One then updates the magnetic field by using a line integral around the cell to get the change in magnetic flux, and then dividing that by the area of the part of the cell outside of the PEC. For the cell marked DM in Fig. 5.8, this would consist of adding up the marked electric fields (two in the x-direction and one in the y-diretion) with appropriate signs. Because of the area divisor, the Dey-Mittra algorithm can be unstable for time steps smaller than the uniform-grid CFL time step limit, and the smaller the area of the cell, the more the time step is limited. In practice, one sets an amount of acceptable reduction of the time step, and then one can compute the fractional cell areas that must be dropped [NCW+09].
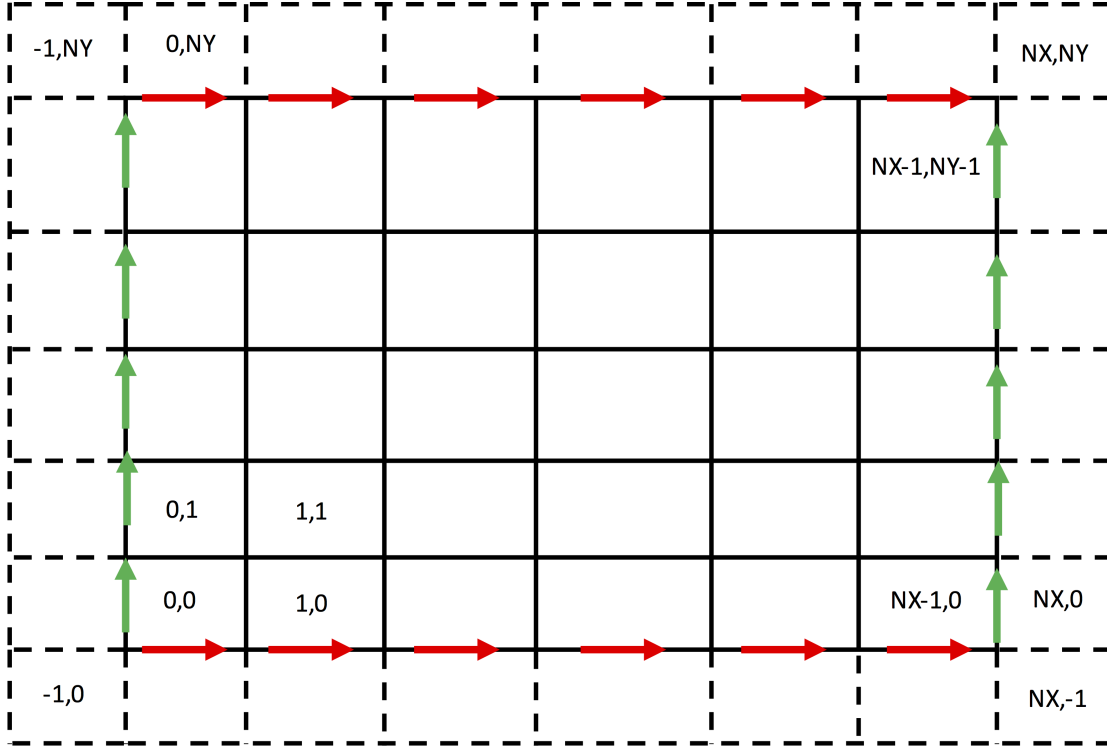
Fig. 5.7: Slab boundary regions for the electric field.

5.4.4 Electrostatics

Electrostatics refers to finding the fields by a different mechanism. The electric and magnetic fields are still present. However, the magnetic field is static and imported, while the electric field at each time step is found by first solving for the potential, which satisfies Poisson's equation,

$$-\nabla \cdot \epsilon(\nabla \phi) = \rho,$$

and then finding the electric field from

$$\vec{E} = -\nabla \phi,$$

which becomes finite differencing in numerics. Because we cannot directly state the solution for the potential, ϕ, but instead we have to solve for the potential, this is an implicit update. Therefore, solving Poisson's equation involves setting up finite difference equations which leads to a system of equations. The system of equations is solved by inverting a matrix at each time step and solving the unknown (ϕ) using the source term (ρ). The default value of the dielectric constant (ϵ) corresponds to that of a vacuum. However, *SpaceTimeFunctions* can be defined to specify regions in which the dielectric constant differs from a vacuum, which can then be inserted into the Poisson Solve.

Electrostatic Slab Boundary Conditions

Poisson's equation, upon discretization connects 2*D+1 grid nodes, as shown by the nodes within the curve in *Conformal boundaries for the electric field.*. Because this *stencil* reaches to each side of the node, it cannot be applied to edge nodes (covered by open circles). On those grids one must apply boundary conditions. For Dirichlet boundary conditions, one specifies the value of the potential on that node.
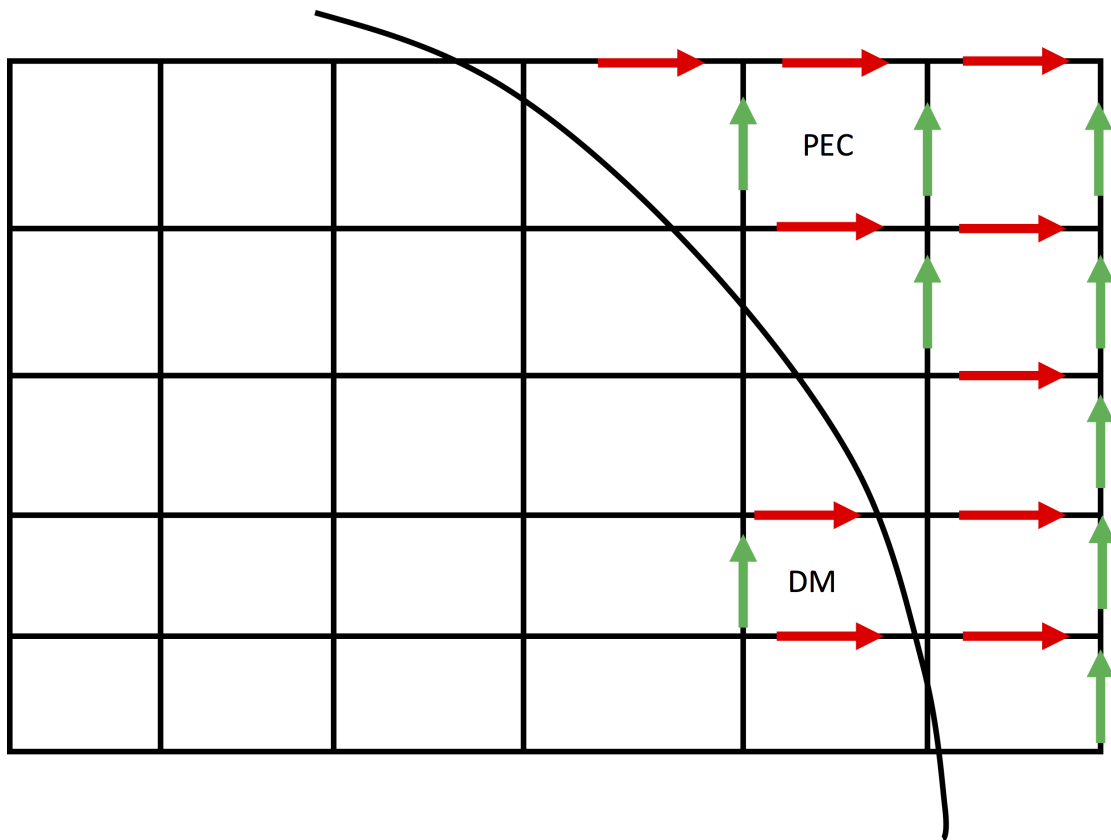
Fig. 5.8: Conformal boundaries for the electric field.

For Neumann boundary conditions, one specifies the value for the difference between the value of the potential on a node and the value on the node just interior. In general one must take care not to specify a mathematically impossible situation. As applied to Neumann boundary conditions, one cannot, e.g., specify zero Neumann boundary conditions on all surfaces while having non-zero net charge in the interior, as that would violated Gauss's law. Similarly, if one has a simulation that is periodic in all directions, consitency requires that it contain no net charge. Further, the matrix is singular unless one sets the value of the potential on at least one node.

Electrostatic Conformal Boundary Conditions

As this is an extensive subject, we simply say that for the nodes interior to a surface of constant potential, one specifies that the potential on that node, rather than being related to nearby points, is simply given. Otherwise one constructs the matrix as before.

Solving Poisson's equation

Upon discretization and applying all boundary conditions, numerically one is left with a large matrix equation to be solved. There are multiple ways to do this within VSim, with direct or iterative solvers. This is discussed in more detail in *Selecting Solvers and Solver Parameters*.

5.5 Particles

Particles can be represented by macroparticles or a fluid. Macroparticles should be used when there is a need to capture kinetic effects. They also have more features in VSim.

5.5.1 Macroparticles

Macroparticles allow one to model kinetics (velocity distributions) for physical particles. Macroparticles are comprised of a certain number of physical particles. The number of physical particles to represent in a single macroparticle is determined by factors such as the physical particle density, volume of a grid cell, and the number of macroparticles in a simulation grid cell. Macroparticles, if used so that there is accurate resolution, produce the same result as would simulating all individual physical particles, but with significantly higher computational speed. We will elaborate more on macroparticle definition and effects on simulation resolution in later sections. For macroparticles there are a number of options for particle loading and emitting from sources, as well as various types of particle sinks available.

There exist over 70 variations in particle type and evaluation, including:

- Boris (Relativistic, Non-relativistic, Tagged, Weighted)
- Electrostatic
- Cylindrical species

For more information on particle species and some of these other algorithms, please visit the Species section of VSim Reference.

Species and their kinds

A common term in VSim for macroparticle is Species, as that is the type of block that defines a set of macroparticles VSim allows you to specify a species kind, determined by the particle type (relativistic, electrostatic, etc.). Within these kinds are options for particles (i.e hydrogen, helium, argon, xenon, etc.) whose charge, mass, cross-section, and other relevant data sets are built into VSim. Alternatively, you can import your own species data in VSim for more custom simulations, so long as the necessary particle information is successfully imported to VSim.

Please visit the section Species Kinds in VSim Reference for all kind options and further details, as well as information on importing your own particle data.

Particle Sources

VSim allows for the implementation of both primary and secondary particle sources, where "primary" refers to initial particles that are introduced into the system, and "secondary" to particles that are created by interactions between these primary particles and either other particles or metal surfaces in the simulation.

Particle "loading" refers to the placement of particles volumetrically in a vacuum, whereas particle "emitting" is the ejection of particles from the surface of a metal.

One can load particles over time in one of the following ways:

- Load all at once. This method is required in electromagnetics simulations for charge conservation.

- Load over about ~1000 time steps

- Load in some custom way using your own external files (.txt files in hdf5 format)

Particle sources in 2D ZR cylindrical coordinates require extra precautions, as the loading algorithm will want to distribute particles uniformly in both directions without accounting for the larger volume/area at large radii that occurs in this coordinate system. To learn how to compensate for this phenomenon, please visit cylindricalparticles in VSim Reference. .. visit Working with Particles in Cylindrical Coordinates in VSim Reference.

Particle Sinks

Particle sinks, on the other hand, generally remove particles from a simulation or from a region therein. There are two basic types of sinks, with many more types of physical sinks available for use in your simulations.

- **Messaging sinks** Automatically established by Vorpal, they are used to communicate particles between processors in parallel runs, enforce periodic boundary conditions, etc.

- **Physical sinks** Physical sinks can remove particles from a region of the simulation, absorb incident particles on a boundary, or even perform more specialized tasks.

Particle sinks typically involve at least upper and lower bounds, and at the most basic level must enclose the entirety of the simulation space that contains particles. If this is not the case and your particles try to travel into regions not included in the simulation grid, Vorpal will likely crash. In the case of periodic boundary conditions, particles "wrap around" from one side of the simulation to the other.

5.5.2 Fluids

Particles can also be represented by fluids. The fluid representation is valid in the limits when the pressure is known. The two cases where this is valid are cold fluids, where the pressure vanishes, and at high collisionality, so that the pressure can be obtained from an equation of state (EOS), such as the adiabatic EOS. The available fluid kinds

- Cold fluid

- Euler fluid

- Neutral Gas

Only the most basic fluid dynamics is supported. There are no conformal boundary conditions nor any internal boundary conditions of any kinds.

5.6 Reactions

Reactions are bulk processes that occur on time scales much shorter than the time step of the simulation. As an example, collisions between atoms or electrons and atoms occur on times of the atomic unit time, 2.4×10^{-17} s, while even in laser-plasma interactions, the laser period is typically of order 3×10^{-15} s. Hence, even on within a time step with the shortest time scales modeled by PIC methods, a collision can be considered an instantaneous process. This is better for plasma discharges that evolve on ms time scales or microwave devices for which the time scale is ns.

The reactions that VSim contains are

- Particle-Particle Collisions

- Particle-Fluid Collisions

- Three-Body Reactions

- Field-Ionization Processes

- Decay Processes

5.6.1 Reactions Implementation

Decay processes are relatively easy, as they involve only a single particle at a time. For particle-particle collisions, VSim uses Direct Simulation Monte Carlo methods, in which one computes the collisions between the pairs within each cell. Three-body reactions are primarily useful for recombination. Finally, field ionization is the creation of an electron-ion pair from a neutral atom in a strong electric field.

Direct Simulation Monte Carlo takes into account that random interactions between particles occur with non-negligible probability only when the particles are in close proximity. To avoid checking the N^2 distances between N interacting macroparticles, the VSim MonteCarloInteractions package limits interactions to those between macroparticles within the same cell. This reduces the number of possible interactions to

$$N_{\text{cells}}(N_{\text{ppc}})^2$$

where N_{cells} is the number of cells in the simulation and N_{ppc} is the number of particles per cell. Hence,

$$N = N_{\text{ppc}}N_{\text{cells}}$$

and

$$N^2 = (N_{\text{cells}})^2 (N_{\text{ppc}})^2 \gg N_{\text{cells}}(N_{\text{ppc}})^2$$

VSim has three reaction frameworks frameworks. The original *reduced* reaction framework had only a limited number of reactions. the *monte carlo* framework has a larger set of reactions. The *reactions* framework is now preferred, as it has an even larger set of reactions as well as using the No-Time-Counter method for algorithmic speedup.

5.6.2 Resolution Issues with Reactions

In order to accurately model the desired physics, one must resolve the physical distributions of the interacting particles and the temporal evolution of the distributions. For sufficient spatial resolution the number of macroparticles in the simulation must be large enough to smoothly resolve the spatial distributions of the species.

As for temporal resolution, each Monte Carlo interaction has an intrinsic time scale set by the physics of the interaction itself. In other words, the probability for an interaction event to occur can be written as $P = dt/T_i$, where dt is the simulation time step and T_i is the natural time scale associated with the interaction itself.

The fundamental probability for an interaction event to occur between two macroparticles in a given cell with volume V in a time step dt is

$$P = \frac{N_1 N_2 \sigma(v) v dt}{N_x V}$$

where v is the relative velocity between the two macroparticles, N_1 and N_2 are the numbers of physical particles per macroparticles in the final-state species. This implies that the natural time scale associated with this numerical process is:

$$T_i = \frac{N_x V}{N_1 N_2 \sigma(v) v}$$

If the simulation time step dt is not small enough to resolve the natural interaction time scale, then inaccurate statistics will result.

The last issue of resolution comes from the need to accurately sample the velocity distributions of the interacting particles. Since the probability for an interaction event to occur non-trivially depends on the particle velocities, and since a single macroparticle samples only one point in velocity space, accurate statistics may only be achieved in some simulations with many macroparticles per cell, such that the velocity distributions of the participating species are well sampled within each cell.

5.7 Histories

A *history* is an array of data that is output at every time step. The type of history that you can include is dependent on the type of simulation (i.e. particle, electrostatic, electromagnetic, etc.) that you are running.

You can incorporate a history in text-setup through one of two ways:

- Including a history by means of a *History* block
- Calling a macro that automatically generates history blocks for you based on a few select input parameters

More details and some example usage of these two methods will be provided later on in *Text Setup*. For further information on history types and parameters as a whole, you can also visit the History section of the *VSim Reference Manual*.

VISUAL SETUP

6.1 Visual-setup Simulations

After you open a new or example simulation that is not text-based, VSimComposer displays the **Setup** window containing an *Editor* pane with a *Simulation Elements Tree*, a *Property Editor*, and a *Geometry View* to allow for easy creation of your simulation. The icon panel remains available on the far left.

Note: A *Navigation Pane* can be shown by clicking and dragging on the vertical bar separating the *Icon panel* from the *Editor* pane. See Fig. 6.1.
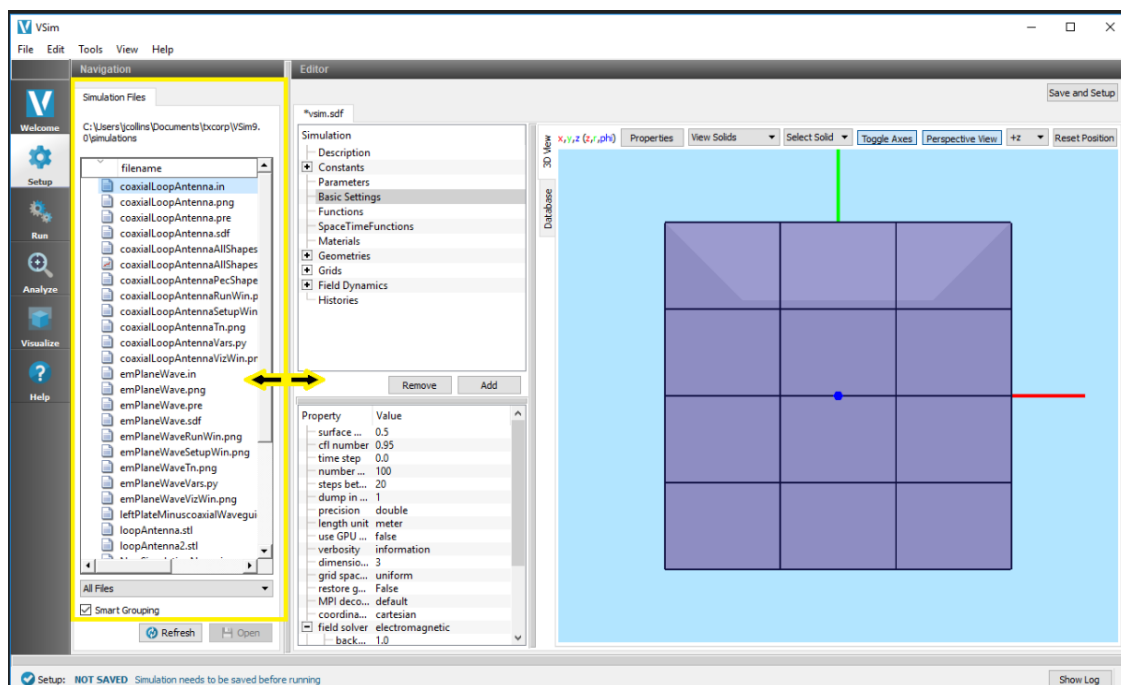
Fig. 6.1: The Navigation Pane

The following sections will go through each of the components of the **Setup** window.

For in depth information on each of the properties and possible values outlined or described in the rest of the chapter, please see VSim Reference: Visual Setup.

The figure *Visual based setup window* illustrates the layout of the VSimComposer **Setup** window using labels for the parts of the interface to which this introduction and the tutorials refer. See Fig. 6.2.
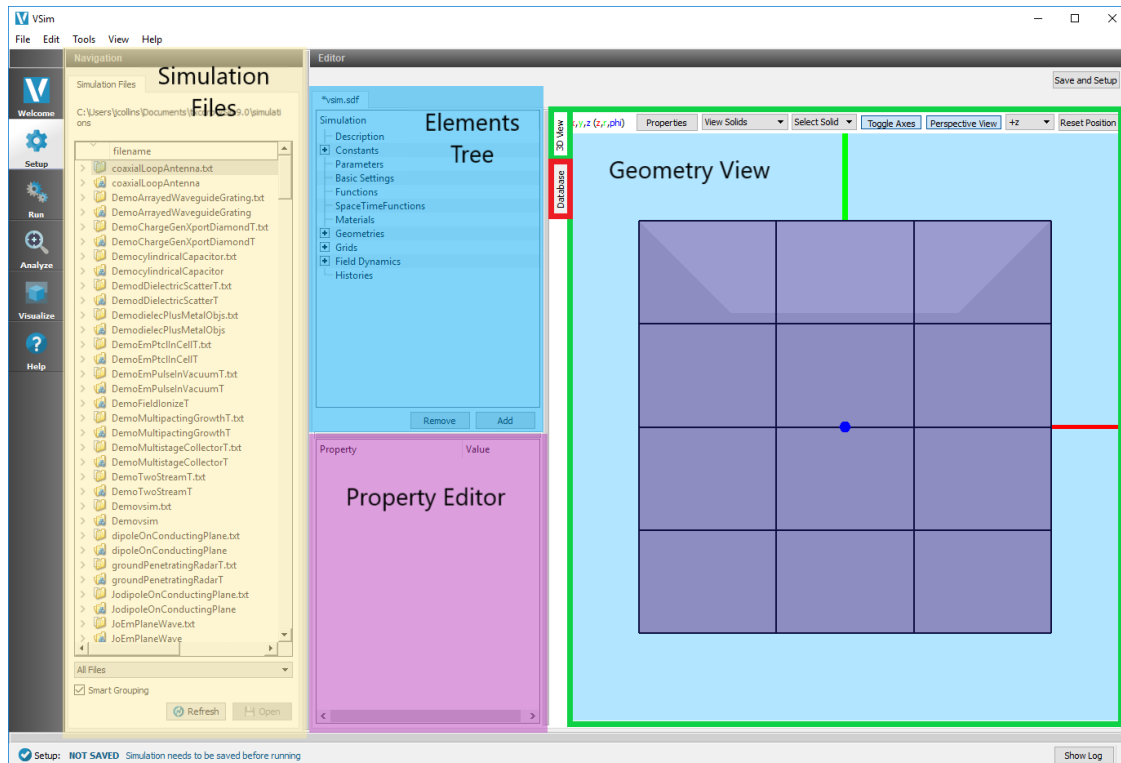
Fig. 6.2: Visual based setup window

6.2 Navigation Pane and Simulation Files

The *Navigation* pane contains a list of *Simulation Files*.

To enable convenient viewing of the list of simulation files, VSimComposer allows you to specify in what order as well as which type(s) of files you would like to view. *Smart Grouping* causes similar types of files to be displayed in the same area of the *Simulation Files* tab list. Turning off *Smart Grouping* causes files to be displayed in alphabetical order rather than by type. *All Files* indicates that you want to see all available files involved in the simulation. You could choose to limit your view to only *Simulation Files*, which are files such as input files and macros that can be edited in the VSimComposer Editor pane, or *Text* files, which include all types of human-readable file formats, or *Data* files, which include incremental dump files and output files that can be visualized.

6.3 Elements Tree Overview

You can navigate through the *Elements Tree* using either your mouse or your keyboard arrows. The up and down arrows will scroll up and down through the list of elements, while the right and left arrows will respectively expand and collapse the elements. To double click, press F2.

Highlighting a particular element in the tree will cause the *Property Editor* to update with Property/Value pairs that can be edited.

New elements can be added under an element in the tree by **Right Clicking**. This will be context sensitive, so for example right clicking *Particle Sources* will open up a tree of all types of available particle sources to choose from. Select one and click on it to add a new instance of that element.

6.3.1 Turn Off/On

This is an option available after **Right Clicking** or using the **Add** button for elements that are not visualized in the 3D View. If it is selected it will gray out the element and it will not be translated into the input file.

This can be particularly useful for comparing results with and without a single element in place without having to go through the process of completely re-specifying it.

6.3.2 Create Clones

Any object can be right clicked and cloned, with all settings set the same as the object. Any number of copies can be created.

6.3.3 Buttons

- Undo

  This will Undo the last thing done. For example it will Undo the addition of an element, or the setting of a property.

- Add Multiple

  This is a shortcut that will add another instance of the last element added to the tree. Note that this can only be used on elements that are not visualized in the 3D view.

- Remove

  This will removed the highlighted element of the tree.

- Add

  This is the equivalent of right clicking a selected Element in the tree, and will present the same options.

We now discuss each of the sections of the Elements Tree.

6.4 Description

The *Description* element holds basic user-supplied text information about the simulation.

6.5 Constants

The *Constants* element contains a set of pre-defined physical constants that can be used in other elements of the simulation. You may also define your own by highlighting *Constants* and either clicking the *Add* button at the lower right of the *Elements Tree* or right-clicking and selecting *Add Constant –> User Defined*.

The name of the user-defined *Constant* can be modified by double clicking on the element in the *Elements Tree* and typing in a new name.

Note: A constant name can contain only alphanumeric characters and underscore and must start with a letter. Our convention is to use ALL_CAPS for constants.

A value can be given to the user defined *Constant* by double clicking on the value in the *Property Editor*. See Fig. 6.3.
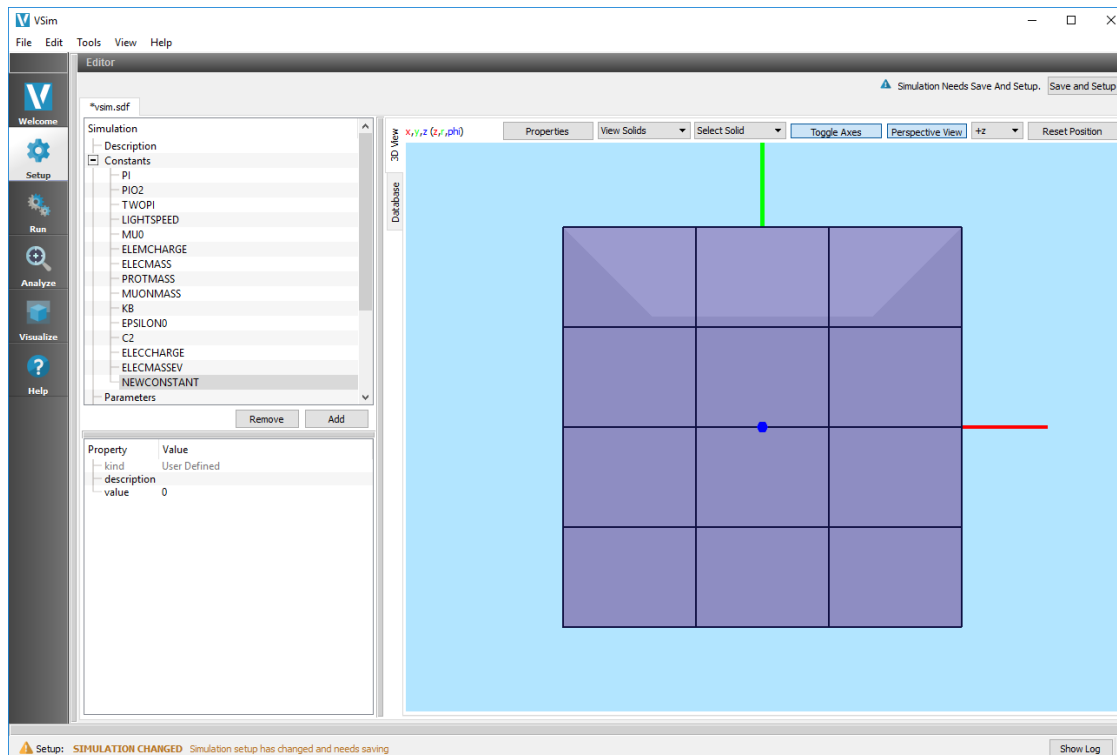
Fig. 6.3: Constant definitions

6.6 Parameters

The *Parameters* element is a location for *evaluated*, user-defined, variables that can be used in other elements of the simulation.

You can add a *parameter* by highlighting *Parameters* and either clicking the *Add* button at the lower right of the *Elements Tree* or right-clicking and selecting *Add Parameter –> User Defined*.

The name of the user-defined *Parameter* can be modified by double clicking on the element in the *Elements Tree* and typing in a new name.

Note: A parameter name can contain only alphanumeric characters and underscore and must start with a letter. Our convention is to use ALL_CAPS for parameters.

An expression can be given to the user defined *Parameter* by double clicking on the expression value in the *Property Editor*. You can use any of the *Constants* as well as any real number in the expression. See Fig. 6.4.

Note: If the expression is not valid, the *parameter* will appear in red in the *Elements Tree*.
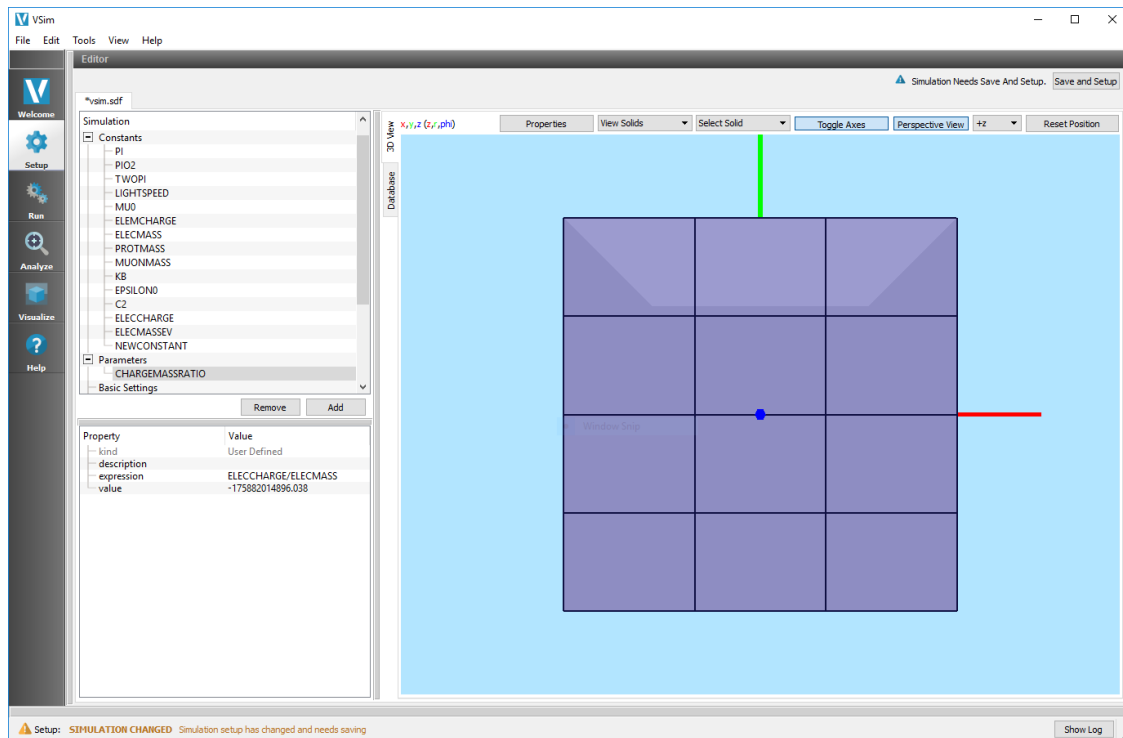
Fig. 6.4: Parameter definitions

6.7 Basic Settings

The *Basic Settings* element contains a group of property/value pairs that define the basic setup of the simulation.

Here you can find properties such as the type of field solve (electromagnetic, electrostatic, prescribed field or no field), the dimensionality (3D, 2D, 1D), whether or not to include kinetic particles in the simulation, and the time step.

6.8 Functions

The *Functions* element is a location for writing user-defined functions that can be used in simplifying the definition of a SpaceTimeFunction. The function can contain any number of arbitrary arguments and is not limited to the default values of x and y.

Note: A *Function* can be used to define another Function or a SpaceTimeFunction. *Functions* cannot be used to define an expression in other elements nor can they be used to define parameters. Expressions are defined by *SpaceTimeFunctions*.

To create your own function, highlight *Function* and either click the *Add* button at the lower right of the *Elements Tree* or right-click and select *Add Function –> User Defined*.

The name of the user-defined *Function* can be modified by double clicking on the element in the *Elements Tree* and typing in a new name.

Note: A function name can contain only alphanumeric characters and underscore and must start with a letter. Our convention is to use lowerCamelCase for function names.

You can define the *Function* by double clicking on the expression value in the *Property Editor*. You can use any of the *Constants*, *Parameters*, or *Functions* previously defined, as well as any real number or Python operator in the expression. See Fig. 6.5.
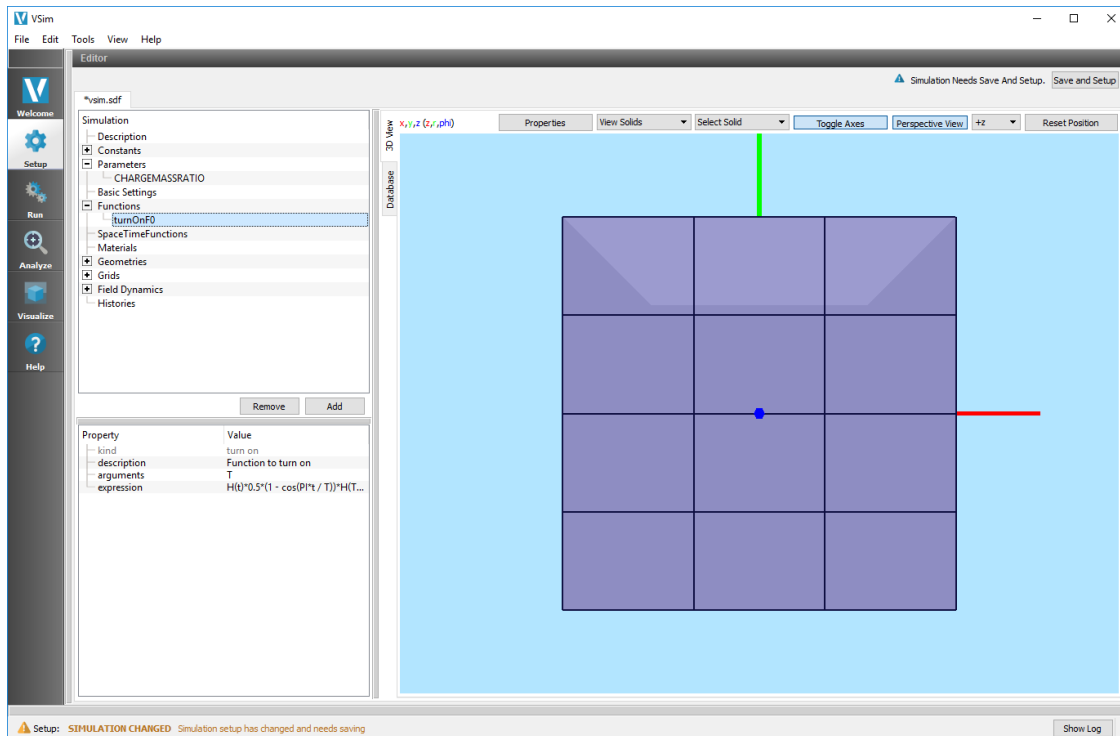
Fig. 6.5: Function definitions

6.9 SpaceTimeFunctions

The *SpaceTimeFunctions* element is a location for writing user-defined functions that specifically depend on the spatial and temporal variables x, y, z, and t. A *SpaceTimeFunction* can be used in other elements of the simulation by right clicking on the value and selecting the defined *SpaceTimeFunction* as shown in the figure *SpaceTimeFunctions definitions*.

To create your own function, highlight *SpaceTimeFunction* and either click the *Add* button at the lower right of the *Elements Tree* or right-click and select *Add SpaceTimeFunction –> User Defined*.

The name of the user-defined *SpaceTimeFunction* can be modified by double clicking on the element in the *Elements Tree* and typing in a new name.

Note: A SpaceTimeFunction name can contain only alphanumeric characters and underscore and must start with a letter. Our convention is to use lowerCamelCase for SpaceTimeFunction names.

You can define the *SpaceTimeFunction* by double clicking on the expression value in the *Property Editor*. You can use any of the *Constants*, *Parameters*, or *Functions* defined above, as well as any real number or Python operator in the expression. See Fig. 6.6.

Note: With SpaceTimeFunctions you may type in the name of a *Constant* or *Parameter* directly, however if you are to change the name of the Constant / Parameter used it will not automatically update to the new name, as it will in other elements of the SDF file.
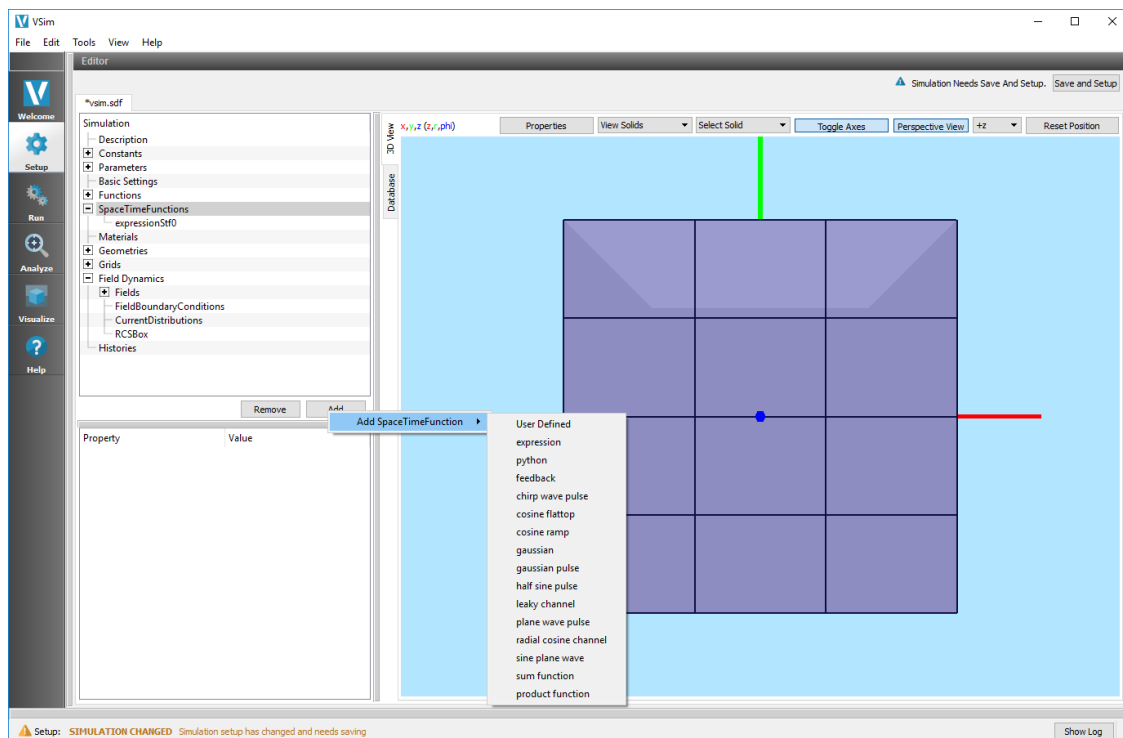
Fig. 6.6: SpaceTimeFunctions definitions

6.10 Materials

The *Materials* element holds information about any materials used in the simulation. There are some *Materials* built into VSim, and the user may import other desired materials.

To import a *Material*, either click the *Add* button at the lower right of the *Elements Tree* or right-click and select *Import Materials*.

The *Materials* file must have the extension .vmat. You can specify your own materials file to import special materials specific to your simulation. See Fig. 6.7.

Note: Tech-X has a standard materials file distributed with VSim. It is located in the data folder of your installation.

Once a material file has been imported into VSimComposer, you can add a specific material to your simulation by highlighting the material of choice and clicking on the *Add To Simulation* button.
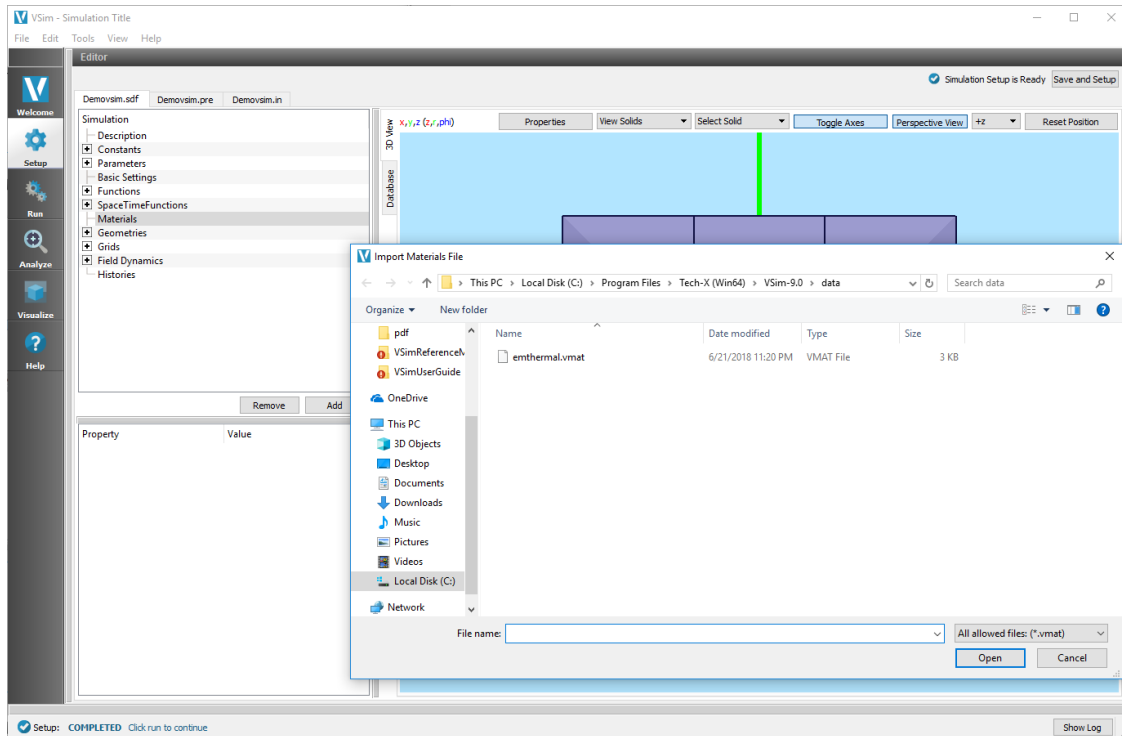
Fig. 6.7: Importing materials

After a material is in the simulation, you can see it under the *Materials* element. The material properties can be modified, if desired. The *Materials* can be assigned to a geometry in the *material* property in the *Properties Editor* pane. See Fig. 6.8.

6.11 Geometries

The *Geometries* element contains information about any geometries that are in the simulation. You can import a file, or create your own with CSG. Geometries present in the simulation can be modified, for example, healed, with holes closed.

Expanding the *Geometries* sub-elements view will show the individual parts (if any) of the imported geometry, or CSG built geometry.

To hide a specific part, uncheck the box next to it.

Note: Hiding a part of the geometry will not remove it from the simulation. VSim will use the full geometry defined in the imported file.

For use in the simulation, a *Geometries* part MUST have a material assigned to it, other wise it is ignored (treated as vacuum).

The material can be assigned to a geometry by double clicking on the material value in the *Properties Editor*. Note that the value of the dielectric assigned to the material, and hence the geometry, is only used if the field solve is electromagnetic. If you are using an electrostatic field solve, the dielectric must be assigned using *SpaceTimeFunctions*. Two examples illustrate the difference in how VSim handles dielectrics in the electromagnetic and electrostatic field
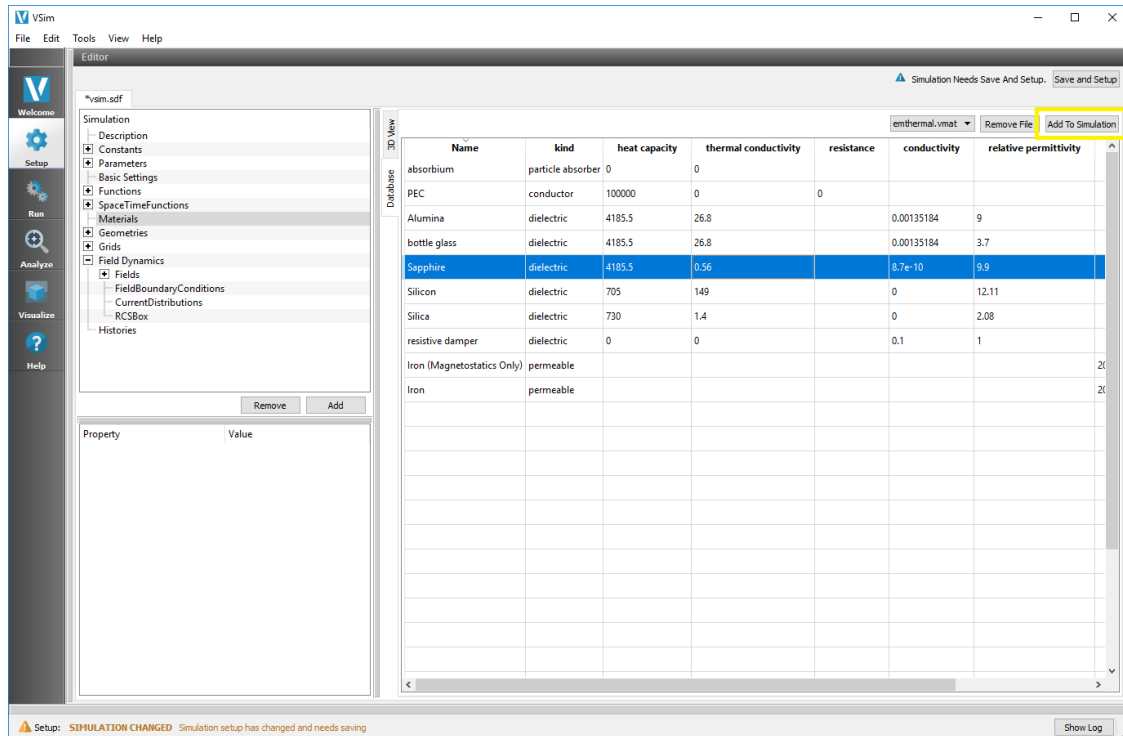
Fig. 6.8: Adding materials to the simulation.

solves: "Dielectric in Electromagnetics" and "Dielectric in Electrostatics" both of which are part of the "VSim for Electromagnetics" examples.

The color of geometry can be set in the *Properties Editor* after selecting the item line of that geometry in the tree. Not that selecting an item is different that selecting the visibility checkbox of that geometry item. Selecting the item means clicking on the line of the item anywhere but at checkbox. Once an geometry item is selected a color property line should appear in the *Properties Editor*. Double click on the color box in the *Value* column to bring up a *Select Color* dialog window to set the color. The transparency control is "alpha" on Windows and Linux (see Fig. 6.9) and "opacity" on Mac (see Fig. 6.10) can also be set in the *Select Color* dialog.

6.11.1 Importing a Pre-defined Geometry

To import a geometry into your simulation, highlight the *Geometries* element in the *Elements Tree* and click on the *Add –> Import Geometries* button located at the bottom of the *Elements Tree*, or simply right click on the *Geometries* element –> *Import Geometries*.

Here you can navigate to a supported file type and open the file. Supported filed types include:

- Step Files (.stp, .step, .p12)
- STereoLithography Files (.stl)
- Visualization Toolkit Files (.vtk)
- Polygon File Format (.ply)
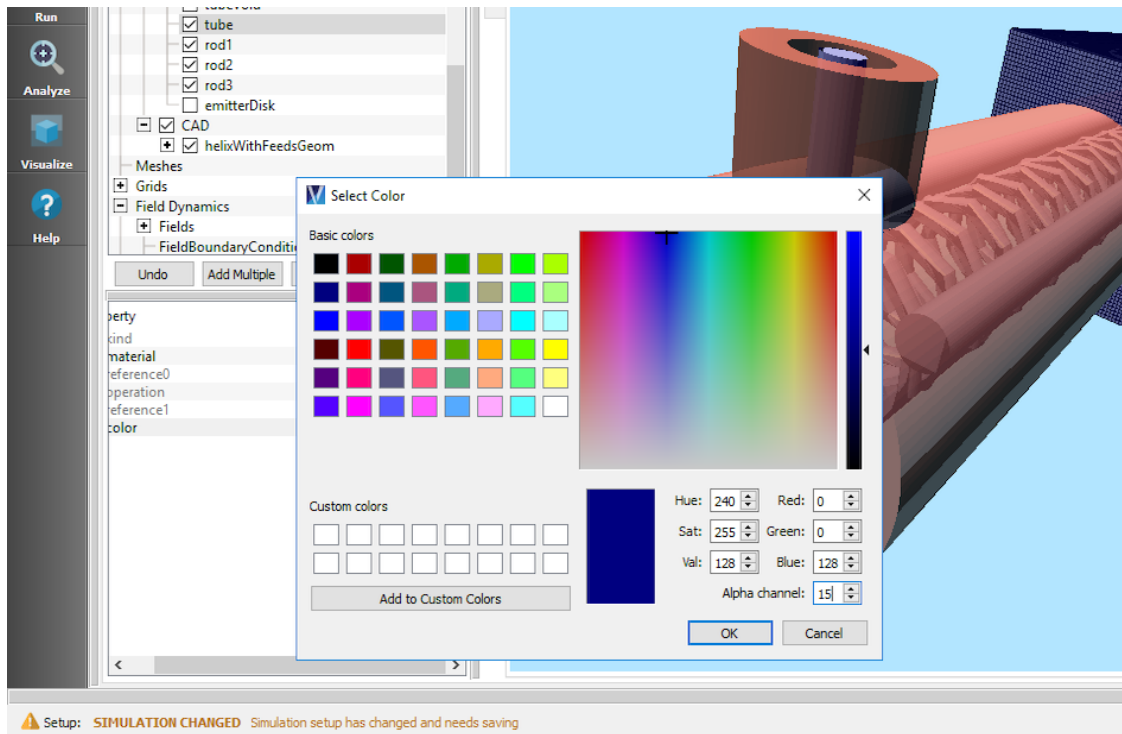
Fig. 6.9: Setting the color property of a geometry through the Select Color dialog on Windows and Linux. The alpha controls the transparency of the selected geometry.
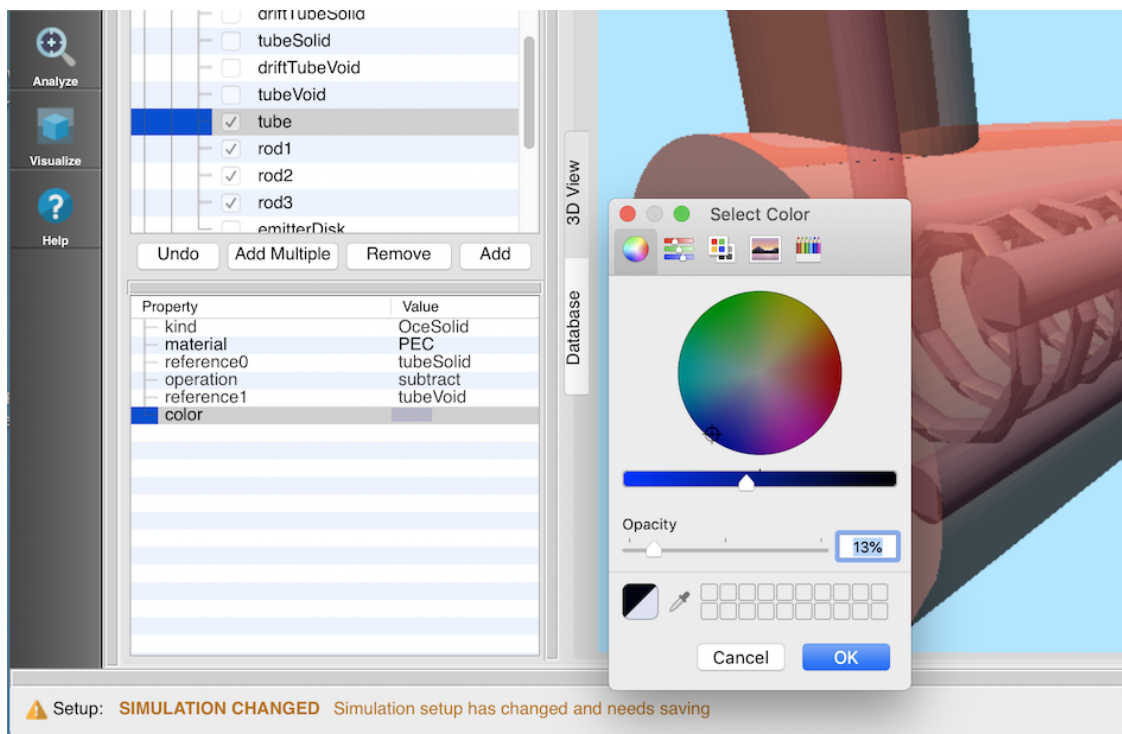
Fig. 6.10: Setting the color property of a geometry through the Select Color dialog on Mac. The opacity controls the transparency of the selected geometry.

6.11.2 Building Your Own Geometry

You can build your own geometry using Constructive Solid Geometry (CSG) to create a complex shape by combining simple shapes using boolean operators.

To do this, highlight the *CSG* element and right click *Add Primitive* and select one of the pre-defined shapes. See Fig. 6.11.
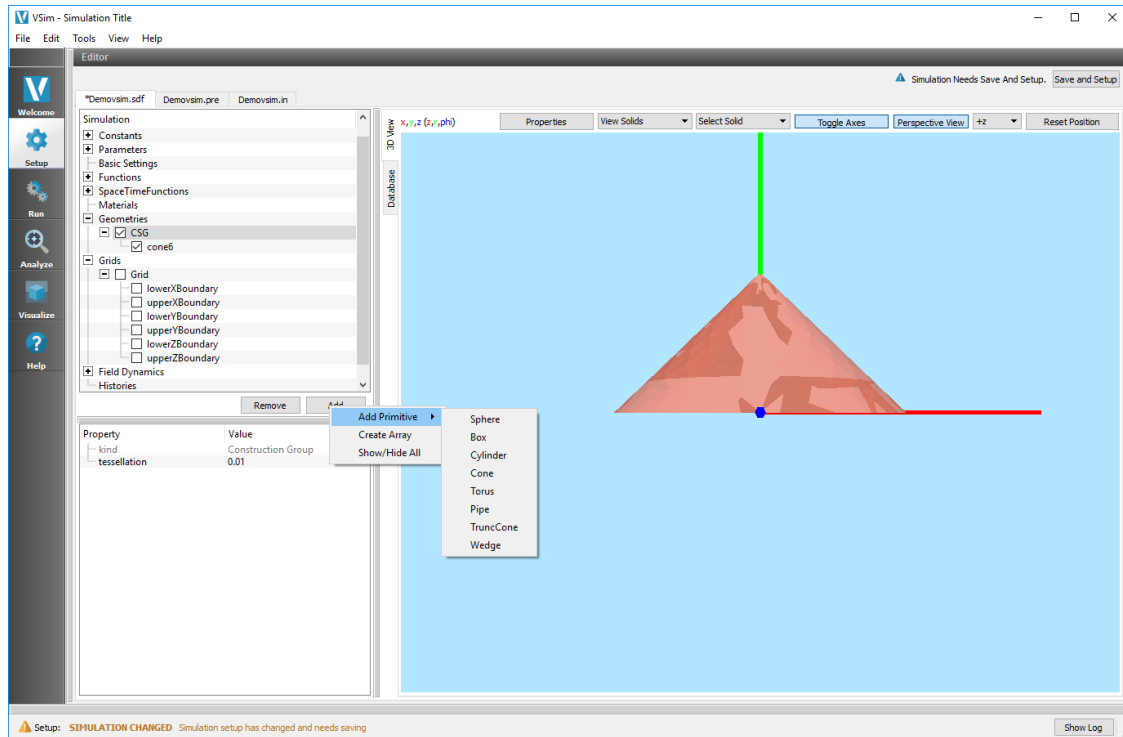
Fig. 6.11: Constructive Solid Geometry (CSG)

After multiple CSG shapes have been added, you can either subtract, union, or intersect them with a boolean operation. This will create a new *Geometries* element.

To do this, highlight the number of shapes you would like to include in the boolean operation, right click, and select the boolean operation you want. See Fig. 6.12. To highlight the second shape on Windows, hold down the 'control' button before selecting the second shape.

Note: The order of highlighting your shapes matters when doing the subtract boolean operation. The second shape will be subtracted from the first. The boolean operation menu will show the operation to be performed based on the order of highlighting.

Note: If a shape is not part of a combined shape through a boolean operation, you can assign a material to it. Once a shape has been combined with another shape, only the combined shape may be assigned a material.
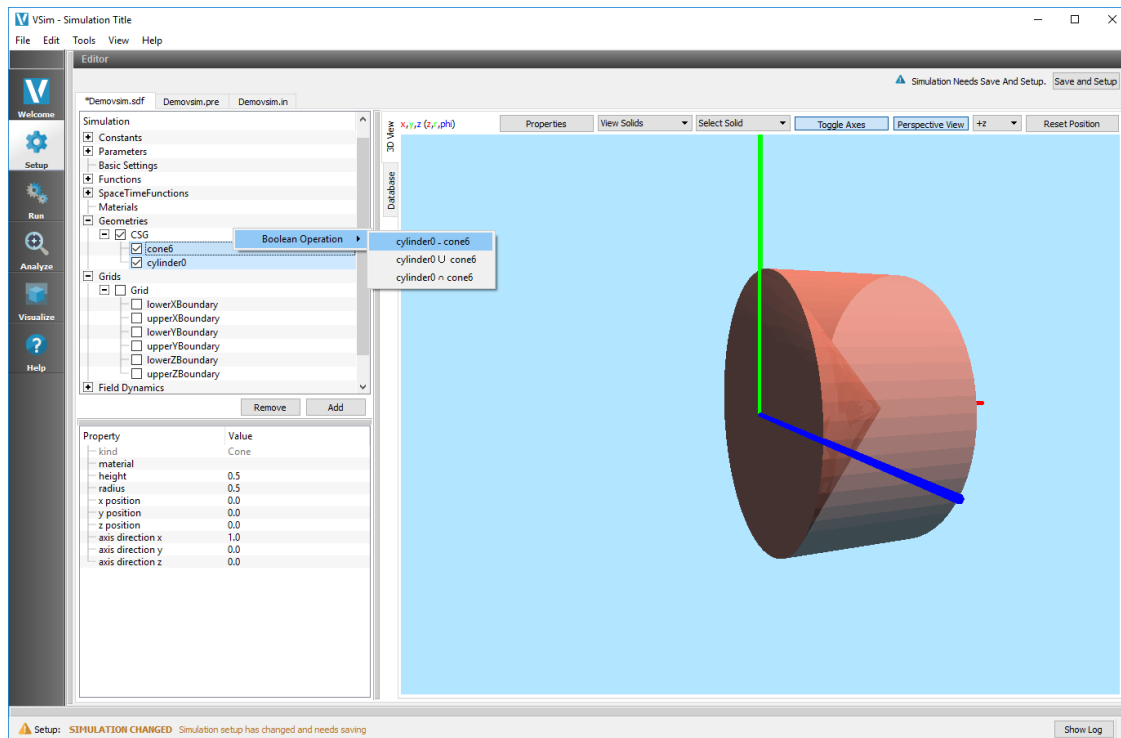
Fig. 6.12: Constructive Solid Geometry (CSG) Boolean Operator

6.11.3 Array Operations

Both CSG and CAD objects may be duplicated with array operations. To do this right click the object in question, and select **Create Array**. This will then open a dialog window to select the *X Count*, *Y Count*, and *Z Count* which will specify the number of objects to create in the respective axial direction. A drop down menu is also available to specify this number with a Constants or Parameter. *X Spacing*, *Y Spacing* and *Z Spacing* will control the spacing between objects - this spacing is in reference to the object in question. Similarly it is possible to assign a Constant or Parameter using the same drop down menu.

There is a Check box available to *Union Created Shapes* which will perform a union boolean operation between all shapes.

This dialog box is shown in the image below

Note: If using a constant or parameter, to define number of array elements/spacing it will hardcode to the value of that constant/parameter at the time of array creation. It will not update the number of elements/spacing should the value of that constant/parameter later change.

6.11.4 Creating Triangulated Surfaces

One can perform many other operations on surfaces in this interface, including analysis of geometries, which involves determining the number of separate surfaces that a geometry contains, and healing of geometries, which involves making them water-tight, closing up any holes. to triangulated form. One can also generate the simulation mesh for the geometries, and visualize it, which allows one to see whether the geometries have fidelity sufficient for a quality simulation.
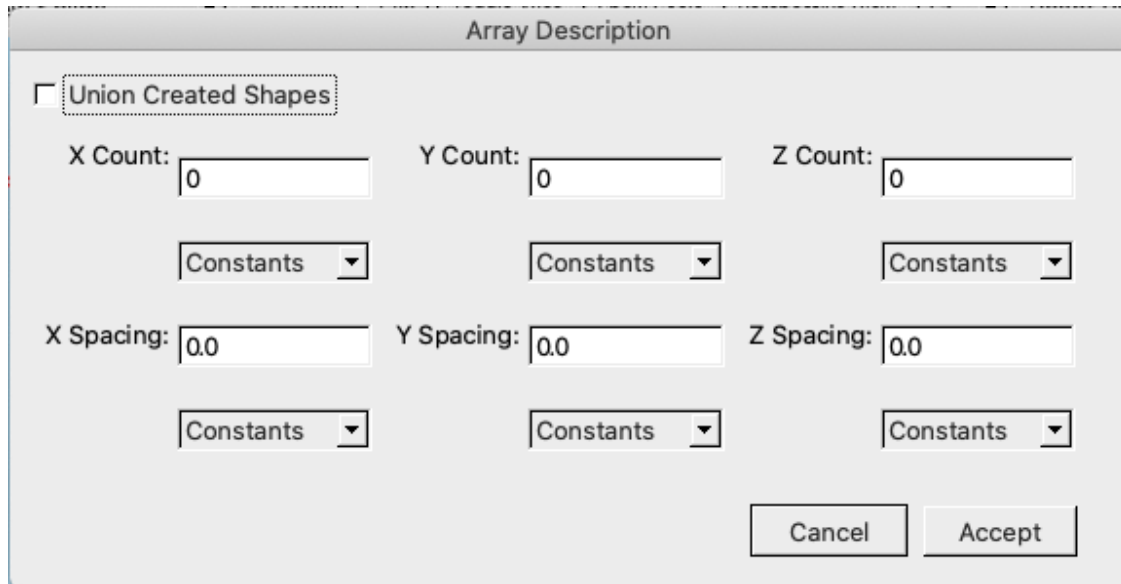
Fig. 6.13: Array Creation

Separating geometries

A common flaw is a CAD file that contains many objects, only some of which are required for the simulation. These objects may also interfere with one another, and may obstruct viewing angles. To begin, one imports the geometry to visually inspect it and see what it contains. One may then use the *Separate Surfaces* command to identify individual objects in the file and extract them into separate files for further use. This is invoked through the menu shown in Fig. 6.14, which comes up through a right click on the geometry.

Selecting *Separate Surfaces* brings up the window shown in Fig. 6.15. Through this interface, one can separate the surfaces. Ultimately, one saves the surfaces for use as separate geometrical items in the simulation. Regardless of what options are chosen in this dialog, the original geometry object will not be changed.

The first step is to "Analyze" the geometry, which will calculate the connectivity between each of the polygons that make up the geometry (triangles, quads, etc). By default, this step will not resolve any ambiguous connectivity - any polygon that could be a part of multiple geometries will be treated as a separate surface. To allow this step to attempt to resolve these ambiguities, one may turn on "Force maximum connectivity". This step may take some time if the geometry is large or particularly complex.

The second step is to identify what types of surfaces to extract. One may choose to extract only closed surfaces, only open surfaces, or both. One may also set a size range for the extracted geometries. Only objects that satisfy all of the given conditions will be extracted.

The final step is to choose how the extracted surfaces will be saved. By default, each surface will be extracted into a new geometry object. You may also choose to group all geometries with the same size, or with the same type, or you may choose to save all surfaces into a single new geometry object. You may also supply an optional suffix to be appended to each of the extracted geometries. Upon being saved, the separate objects will show up in the Geometries tree.

For complex geometries, it may be desirable to use *Separate Surfaces* multiple times. For example, one may first choose to extract all geometries that are "open", i.e. geometries that must be repaired before use. One may then choose to extract only the largest of these geometries, discarding any objects that consist of only one or two triangles. Finally, one may save each of these large geometries into separate files for further analysis and repair.
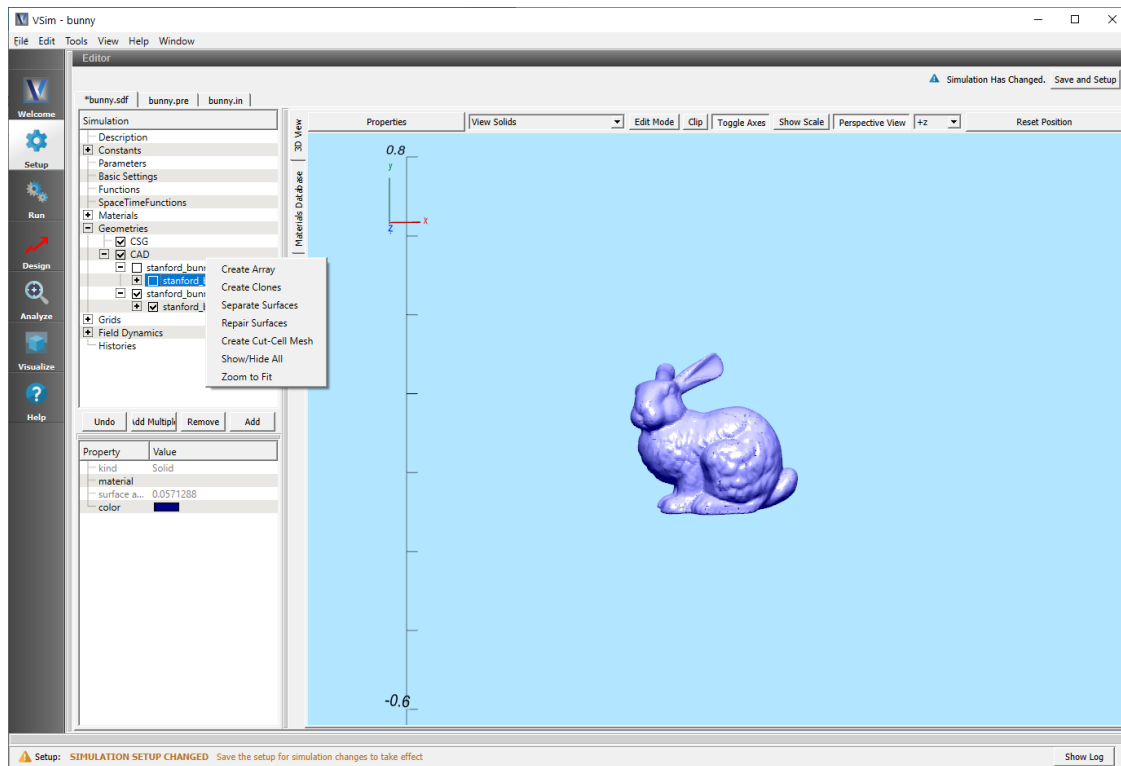
Fig. 6.14: Geometry analysis menu.

Healing geometries

Imported geometries may also have damage, such as holes, that will negatively affect the simulation. To detect and repair such geometry issues, one may use the *Repair Surfaces* command shown in Fig. 6.14. A dialog will appear that will guide you through the process. Again, regardless of what options are chosen in this dialog, the original geometry object will not be changed.

The first step is to "Analyze" the geometry, which will calculate the connectivity between each of the polygons that make up the geometry (triangles, quads, etc). By default, this step will not resolve any ambiguous connectivity - any polygon that could be a part of multiple geometries will be treated as a separate surface. To allow this step to attempt to resolve these ambiguities, one may turn on "Force maximum connectivity". This step may take some time if the geometry is large or particularly complex.

When analysis has completed, the dialog will display a list of all holes (if any) in the target geometry. The "Edge Count" value is the number of edges that border the hole. A hole with three edges is a triangle, four edges is a quad, and more edges that that is a larger polygon. The "Display" button will rotate and zoom the view in an attempt to present the given hole to the camera. Note that this may not be successful if the hole is topologically complex or hidden by another surface. The "Repair" button allows you to fill the hole with new triangles. You may then also undo the operation if the repair is unsatisfactory. Upon conclusion of Repair, the patch needed for the repair shows up in the tree.

To finalize the process, press the "Save" button. Another dialog will appear that will allow you to choose how to save the results. "Export Healed Geometry" will create a new geometry object that includes the original geometry and any newly generated triangles. "Export Patch" will create a new geometry object that contains only the newly generated triangles. The healed object will then show up in the tree for use in the simulation.
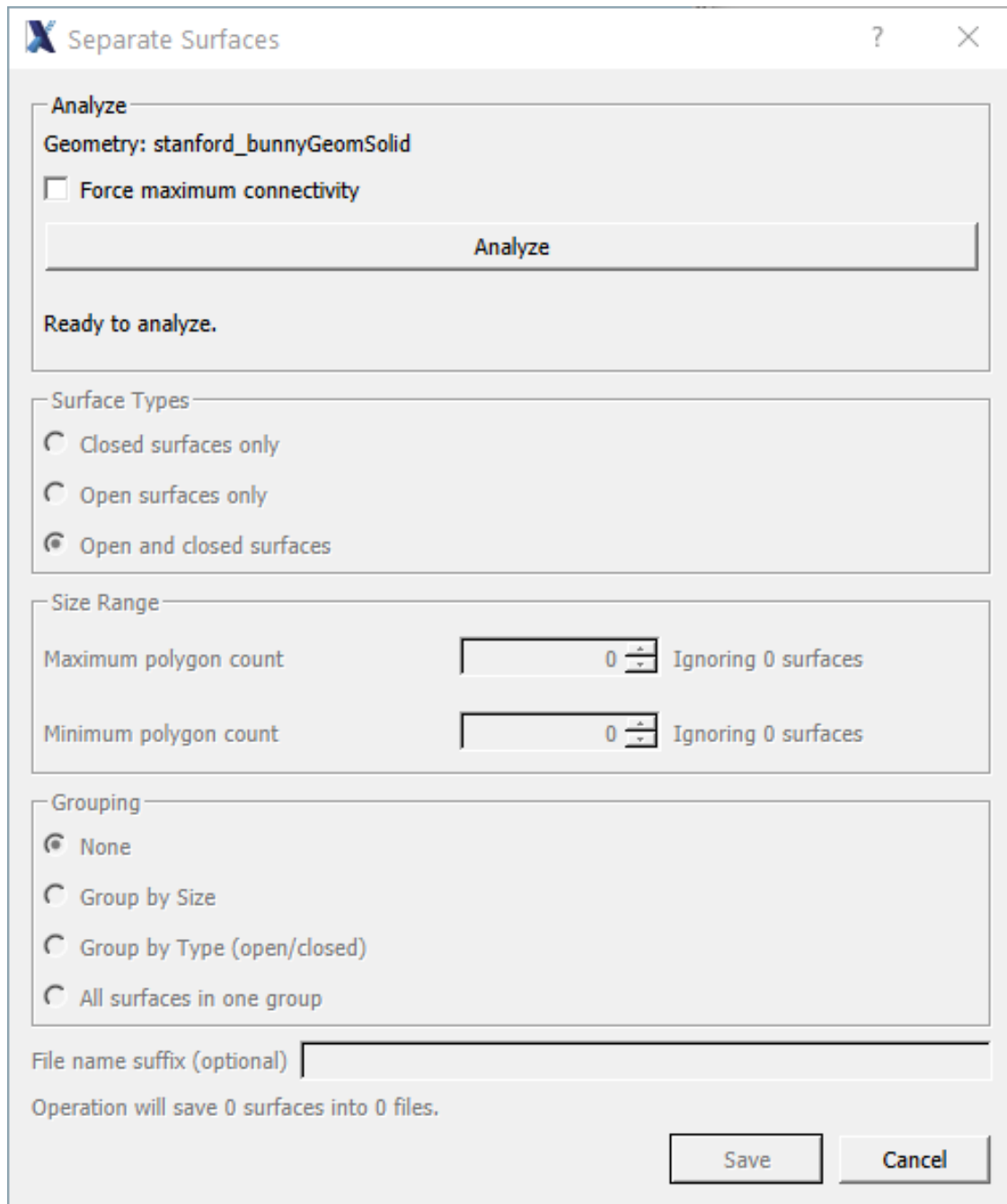
Fig. 6.15: Window for separating surfaces.
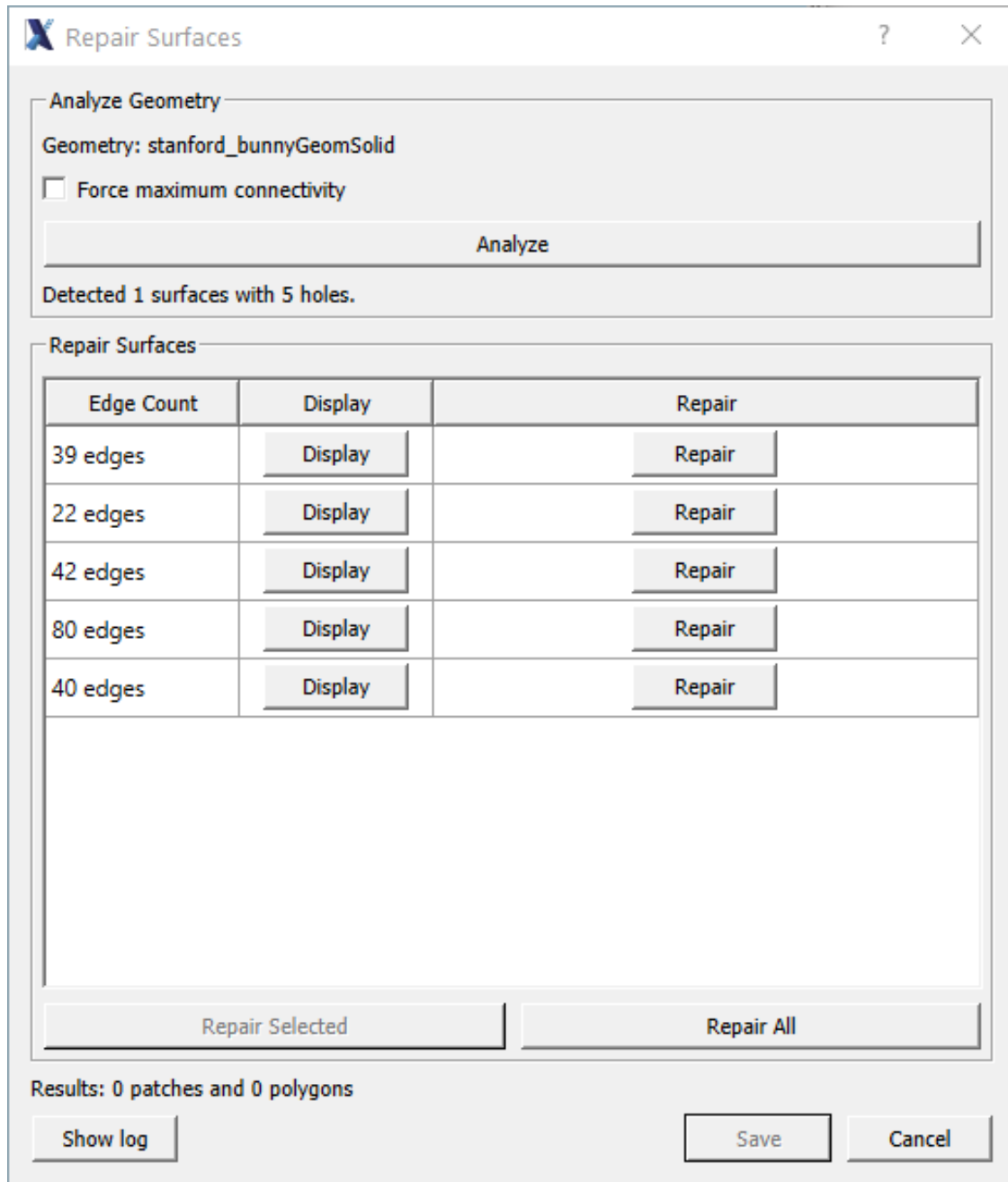
Fig. 6.16: Window for repair surfaces.

Generating Simulation (Cut-Cell) Meshes

Once a shape has been created using a primitive or imported, the shape will appear under the "CSG" tab if you created the shape or "CAD" tab if you imported the shape. If you want to view the mesh that is used to represent that shape in the simulation, then right click on the shape and click on *Create Cut-Cell Mesh*. The mesh will be added under the "Meshes" tab in the Visualization pane.. If you uncheck the box next to the shape under the "Geometries" tab, this will hide the solid shape, revealing the surface meshing used in the simulation. If the mesh is not revealed, check the box next to the shape under the "Meshes" tab. The resolution of the surface mesh depends on the resolution of the grid, which is determined under the "Grids" tab. Increasing "xCells" makes both the grid and surface mesh finer in the x-direction.

6.12 Grids

The type of grid is determined in the *Basic Settings* element. Its parameters are determined in the *Grid* element.

By default, a uniform Cartesian grid is added to your simulation with dimensions of 1m x 1m x 1m and cell numbers of 3, 4, and 5 in x, y, and z respectively.

To modify the type of grid, change the *Basic Settings* properties *coordinate system*, *dimensionality*, and *grid spacing*. See Fig. 6.17.
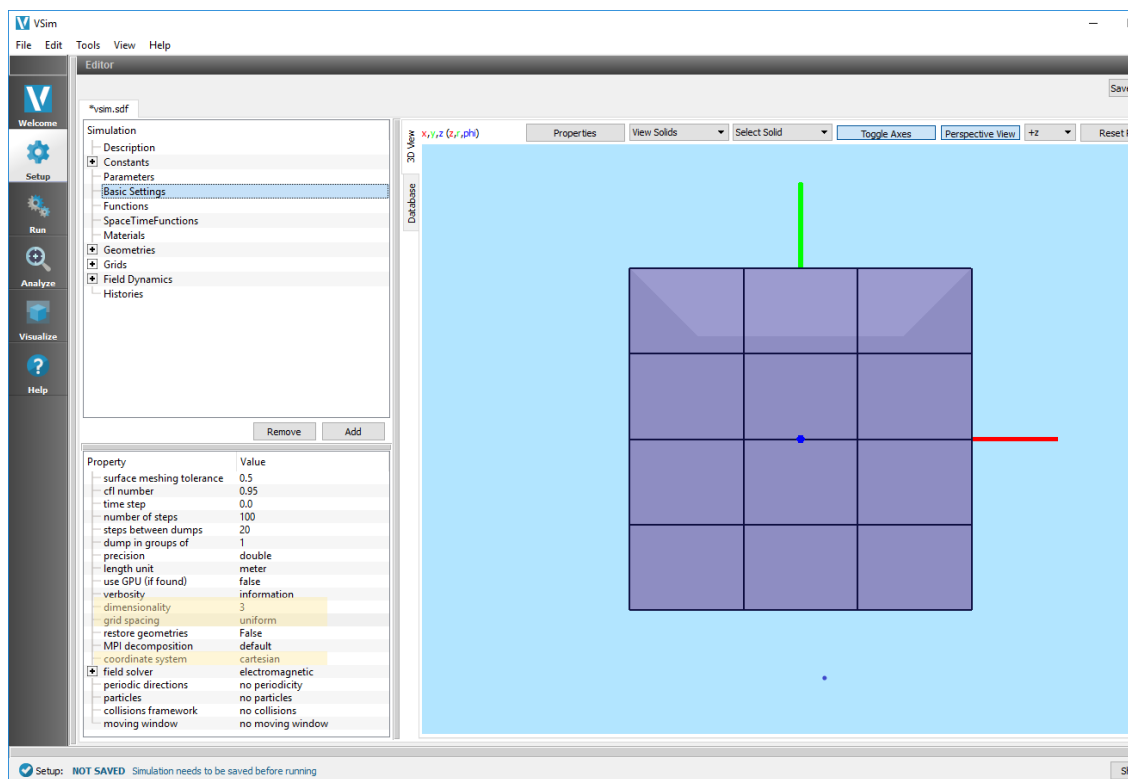
Fig. 6.17: Grid choices

The size of the domain can be set using the *Min* and *Max* properties of the *Grid* element. The number of cells in each direction can also be specified. See Fig. 6.18.

Note: Only one grid may be added to any one simulation at a time.
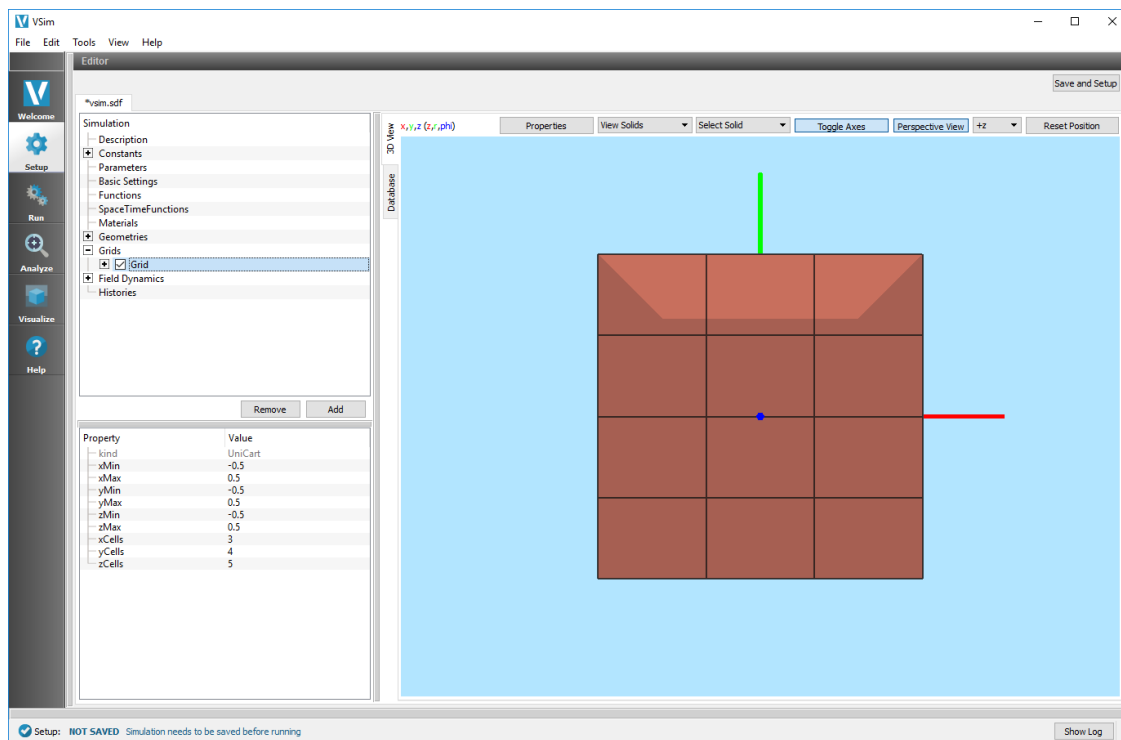
Fig. 6.18: Grid settings

Note: A grid can be resized to fit the bounds of a geometry by right clicking on the Grid element and choosing *Resize Grid*. Resizing the grid puts in numbers, replacing amu constants and parameters.

6.13 Field Dynamics

The type of *field solver* is determined in the *Basic Settings* element. Its parameters are specified in the *Field Dynamics* element.

6.13.1 Field Solver

4 field solvers are available in VSim

Electromagnetic Solver

The electromagnetic solver will solve the E, B and if explicitly included (or particles are in the simulation) a J field is added to the simulation.

Electrostatic Solver

The electrostatic solver will solve an E, Phi and Charge Density, with the option to include external magnetic fields.

Prescribed Fields Solver

The prescribed field solver can be used to import previously solved external fields, and then provide a time dependent variation. This can be used when studying particle dynamics, particularly in multipacting simulations.

No Field Solver

A no field solver is used if only Particle Dynamics are to be studied, without the impact of fields.

6.13.2 Fields

Depending on the type of solver chosen, default fields will be initialized in the simulation. For an electrostatic simulation, default fields will be *Phi*, *Charge Density* and *Electric Field*. You can optionally add a *Background Charge Density* field or *External Field* by clicking the *Add –> Add Field* button located at the bottom of the *Elements Tree*, or simply right clicking on the *Fields* element *–> Add Field*.

An *External Field* is used for importing a *Magnetic* field in electrostatic simulations with particles.

For an electromagetic simulation, default fields will be *Electric Field* and *Magnetic Field*. You can optionally add a *Current Density* field or *External Field* by clicking the *Add –> Add Field* button located at the bottom of the *Elements Tree*, or simply right clicking on the *Fields* element *–> Add Field*.

An *External Field* in electromagetic simulations can be a *Magnetic*, *Electric*, or *Current* field. External fields are used to effect particle movements in simulations.

Field Initial Conditions

An initial condition can be added to any field by clicking the *Add –> Add FieldInitialCondition* button located at the bottom of the *Elements Tree*, or simply right clicking on the particular *Field* element *–> Add FieldInitialCondition*. See Fig. 6.19.

Field Boundary Conditions

A boundary condition can be added to any field by clicking the *Add –> Add FieldBoundaryCondition* button located at the bottom of the *Elements Tree*, or simply right clicking on the particular *Field* element *–> Add FieldBoundaryCondition*.

Current Distributions

A current distribution can be added to any field by clicking the *Add –> Add CurrentDistribution* button located at the bottom of the *Elements Tree*, or simply right clicking on the particular *Field* element *–> Add CurrentDistribution*.

A distributedCurrent is a volume current source where you can provide the min and max values in each direction. A distributedCurrent will show up on the *Geometry View*. You can hide a distributedCurrent by unchecking the box next to it. Hiding a current will not remove it from the simulation, it'll just hide it from the geometry view. See Fig. 6.20.

Poisson Solver

The type of Poisson solve and any preconditioner can be set under the *solver* and *preconditioner* properties of the *Properties Editor*.
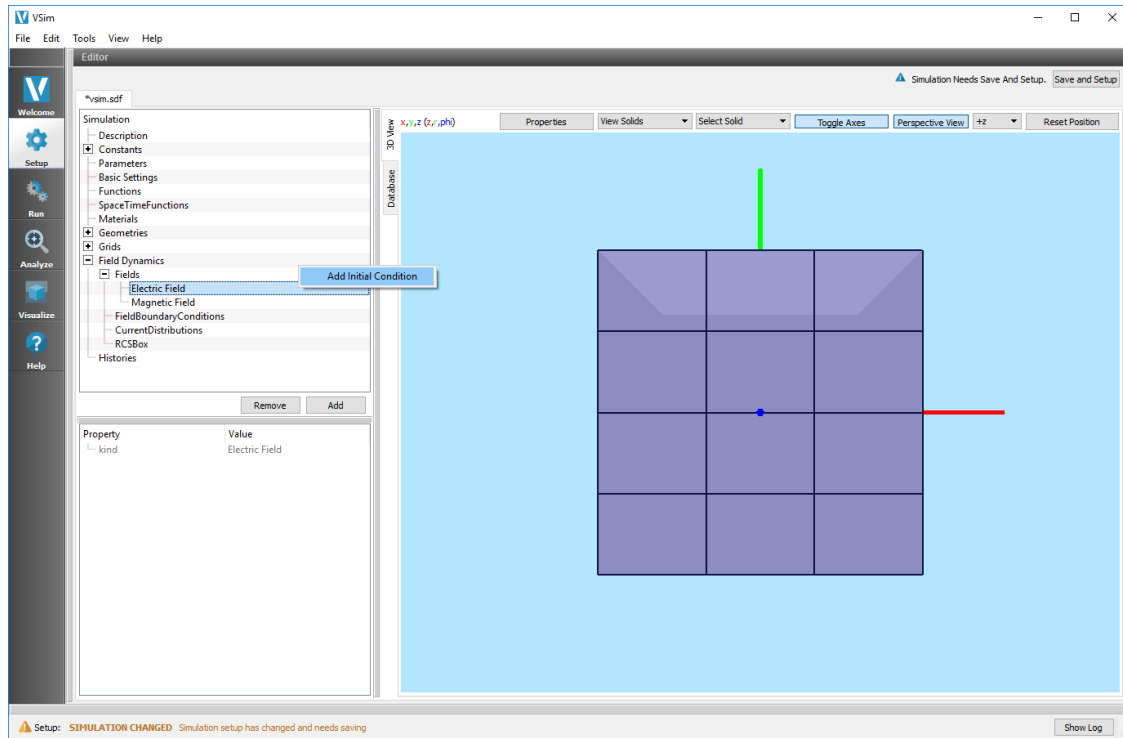
Fig. 6.19: Adding an initial condition to a field
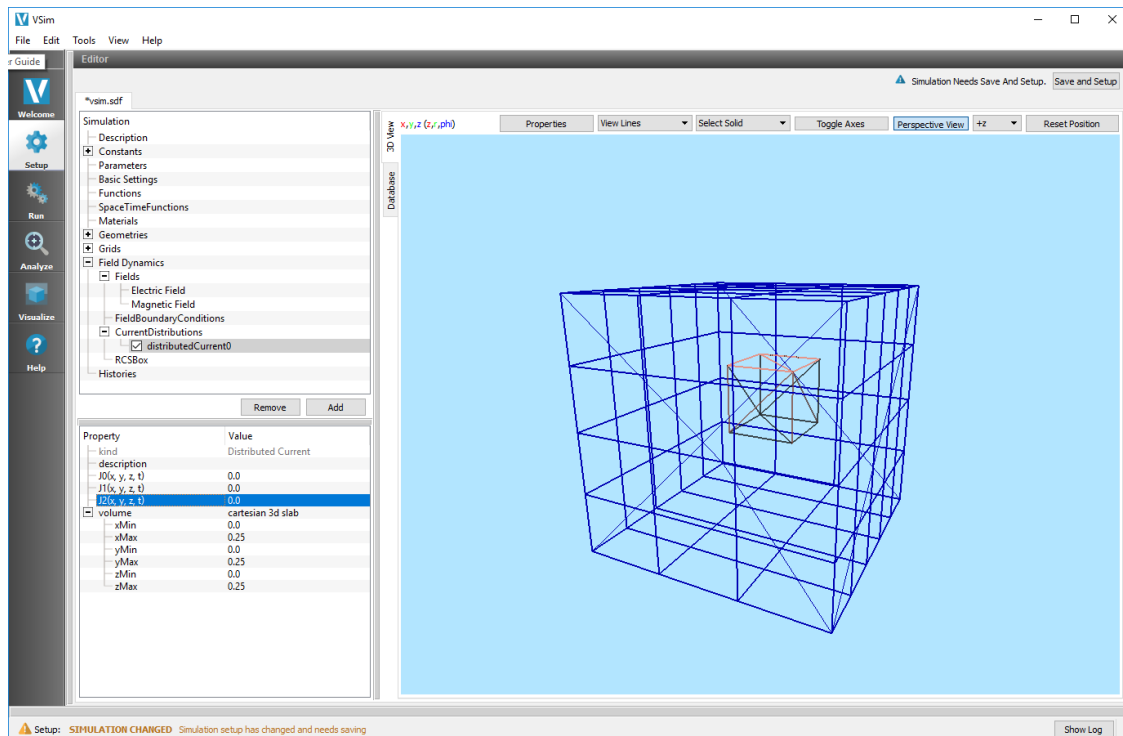
Fig. 6.20: A volume of current is shown in the smaller/interior box. The outer box is the grid.

Note: Changing the solver type may introduce more properties due to the context-sensitive nature of the input.

6.14 Particle Dynamics

The inclusion of *Particle Dynamics* is determined by the value of *particles* in the *Basic Settings* element. If *particles* is set to *no particles*, then no particles are in the simulation and the *Particle Dynamics* element is hidden.

If *particles* is set to *include particles* then the *Particle Dynamics* element is shown and futher properties can be set.

The *Particle Dynamics* element holds information on any kinetic particles, background gases, and collisions in the simulation.

6.14.1 KineticParticles

Electrons, charged particles, and neutral particles can be added to the KineticParticles element.

To add kinetic particles, click the *Add –> KineticParticle* button located at the bottom of the *Elements Tree*, or simply right click on the *KineticParticles* element and select *Add KineticParticle* and then choose the type of particle you want to include.

The properties of each kind of *KineticParticle* are modifiable in the *Properties Editor* pane.

6.14.2 Fluids

Neutral Fluids can be added to the simulation. To add a fluid, right click the Fluids button and select *Add Fluid*

A fluid is a volume distribution where you can provide the min and max values in each direction. A fluid will show up on the *3D View*. You can hide a fluid by unchecking the box next to it. Hiding a fluid will not remove it from the simulation, just hide it in the 3D view. See Fig. 6.21.

6.14.3 Collisions

There are three frameworks for setting up particle collisions available in VSim. The newest, most flexible, and fastest is the *Reactions* framework. The *Reactions* framework supplants the *Monte Carlo Interactions* framework. The *Impact Collider* (called "ReducedCollisions" in the Visual Setup) framework is the oldest framework, and is limited to interactions between kinetic particles with a neutral fluid, but runs very quickly.

In the Visual Setup, only one framework can be used at a time.

Reactions

To include Reactions in the Visual Setup, first select the *Basic Settings* element of the setup tree and ensure that the *particles* dropdown menu is set to "include particles" and the *collisions framework* dropdown is set to "reactions."

With these settings selected, collisions can now be set up within the *Particle Dynamics* element of the setup tree. The Reactions are organized between five options: Particle Particle Collisions, Particle Fluid Collisions, Three Body Reactions, Field Ionization Processes, and Decay Processes. By highlighting one of these five options, right clicking, and adding a collision process a user is setting a `RxnProductGenerator`(see :ref:`VSim Reference Manual: Text Setup: Reactions <rxn-rxnProductGenerator>` for more information on `RxnProductGenerators`).
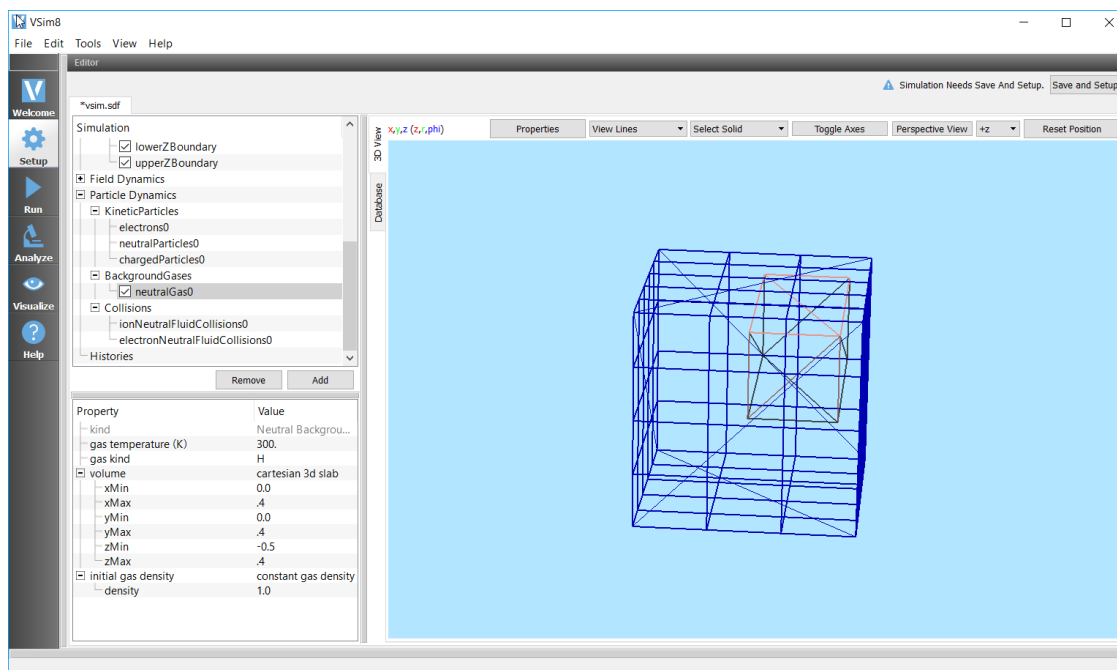
Fig. 6.21: A volume of fluid is shown in the smaller/interior box. The outer box is the grid.

Note: The distinction between the Particle Particle Collisions and Particle Fluid Collisions is artificial in the visual setup and is made for the convenience of the user.

After adding a collision process to the tree, the user then selects the reacting species, cross-sections/reaction rates, and other reaction attributes. The species and fluids in the drop down menu for reactants and products are limited such that only selections appropriate for the process are available. Charge and mass conservation is checked during the translation from `.sdf` to `.in`. If there is a charge or mass violation, an error will be thrown.

If the drop-down menu used to set the interacting particle species is empty, make sure you've added the necessary KineticParticle or BackgroundGas for the type of collision.

Reduced Collisions (Impact Collider)

To include Reduced Collisions in the Visual Setup, first select the *Basic Settings* element of the setup tree and ensure that the *particles* dropdown menu is set to "include particles" and the *collisions framework* dropdown is set to "reduced". With these settings selected, collisions can now be set up within the *Particle Dynamics* element of the setup tree.

To add collisions between kinetic particles and a neutral background gas (fluid), click the *Add –> Add ParticleFluidCollision* button located at the bottom of the *Elements Tree*, or simply right click on the *Collisions* element and select *Add ParticleFluidCollision* and choose whether either *Electron Neutral Fluid Collision* or *Ion Neutral Fluid Collision*. After making a selection, a new element will appear in the tree. Choose the particle species and the background gas that will interact.

To add a specific collision process, highlight this new element and choose a specific collision process from the *Add CollisionProcess* menu which will appear next to the mouse arrow. When a specific collision process is added, a new element will appear. The cross-sections for the interaction process will be set in this element.

See Fig. 6.22 for an example of adding collisions to a simulation in the Visual Setup.
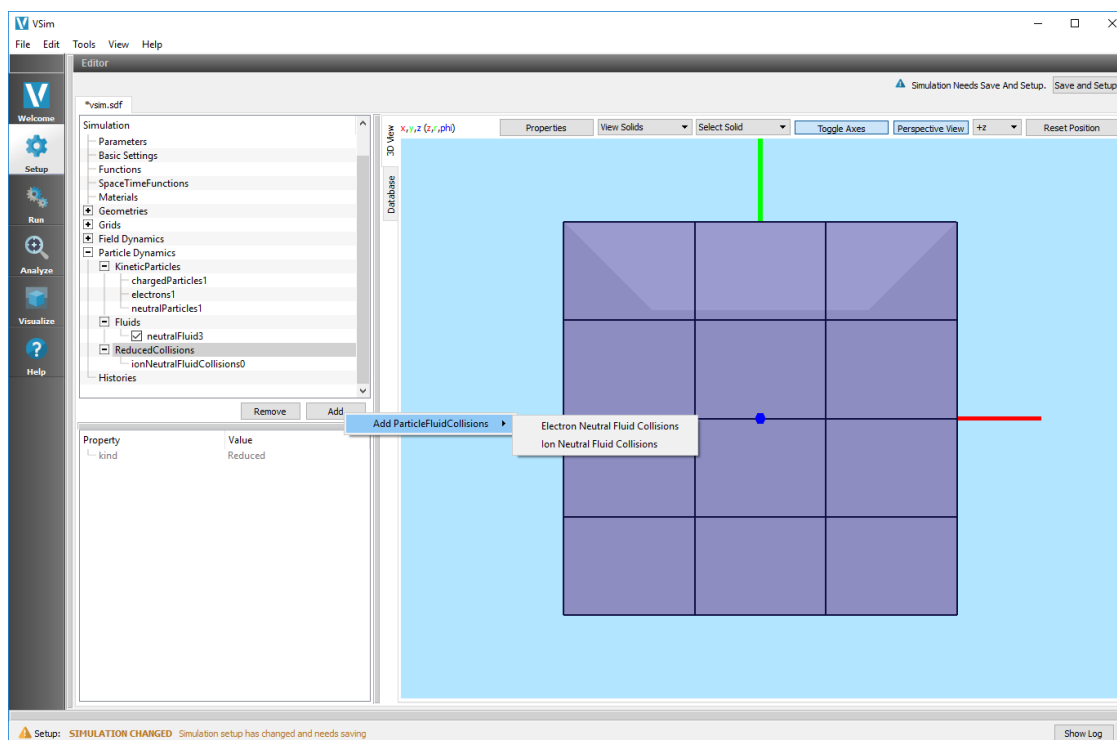
Fig. 6.22: Collisions

Monte Carlo Interactions

To include Monte Carlo Interactions in the Visual Setup, first select the *Basic Settings* element of the setup tree and ensure that the *particles* dropdown menu is set to "include particles" and the *collisions framework* dropdown is set to "monte carlo".

With these settings selected, collisions can now be set up within the *Particle Dynamics* element of the setup tree. The Monte Carlo are organized between five options: Particle Particle Collisions, Particle Fluid Collisions, Three Body Reactions, Field Ionization Processes, and Decay Processes. The user can add a specific interaction process by highlighting one of these five options, right clicking, and selecting a process from the *Add CollisionType* menu.

After adding a collision process to the tree, the user then selects the reacting species, cross-sections/reaction rates, and other reaction attributes.

6.15 Histories

Histories are used to calculate and record data about fields and particles in a simulation.

6.15.1 ArrayHistory

An *Array History* will output an array of data for each time-step.

Possible *Array Histories* include:

- Far-Field Box Data
- Field Slab Data

- Particle Momentum

6.15.2 ComboHistory

Combo Histories are used to do operations on other histories. The operation is done at every time step and the resulting values are recorded as a new history. The output will be a 1D array of the value vs time.

Possible *Combo Histories* include:

- Binary Combination History

6.15.3 FieldHistory

Field Histories record on a per time-step basis. Field histories are used to measure quantities such as the value or energy of the field at a location. The output will be a 1D array of the value vs time.

Possible *Field Histories* include:

- Accelerating Voltage
- Electric Field Energy
- EM Field Energy
- EM Field on Plane
- Magnetic Field Energy
- Field at Position
- Poynting Flux
- Pseudo-potential

6.15.4 LogHistory

A *Log History* will record data on a per-event basis rather than at each time step. For instance, an *Absorbed Particle Log* will record information about each and every particle that strikes a chosen absorbing surface. The output will be a 1D array of the value.

Possible *Log Histories* include:

- Absorbed Particle Log
- Emitted Particle Log

6.15.5 ParticleHistory

Particle Histories record on a per time-step basis. Particle histories are used to measure quantities such as the total number of particles in a simulation at each step, or the current absorbed at a chosen absorbing surface at each step. The output will be a 1D array of the value vs time.

Possible *Particle Histories* include:

- Absorbed Particle Current
- Absorbed Particle Energy
- Emitted Current

- Number of Macroparticles
- Number of Physical Particles
- Particle Energy
- Particle Energy Change from Boundary

6.15.6 ComboHistory

A combo history is used to combine other histories, two types are available:

- Combination History
- Time Average

6.16 Property Editor

The *Property Editor* allows for setting of specific properties under each of the *Elements* from the *Elements Tree* for the simulation. Such properties might include sizes in the X, Y, and Z directions, the type of particles, and specifics of a field solve.

The *Property* and *Value* inputs are context-sensitive. The availability of a particular property may depend on other properties or selections of the *Element Tree*. For instance, changing the solver value in the *PoissonSolver* element will bring up a new set of solver *properties* to be set. Refer to the figure *Constructive Solid Geometry (CSG) Boolean Operator* (Fig. 6.12) to see the same principle with regard to geometry options.

6.17 Geometry View

The *3D View* section can be used to view the simulation setup including the geometry, grid, and source.

6.17.1 Buttons

- Properties

  This allows for selection of the line width and point size, as well as how Colors will be assigned to shapes in the *3D View* See Fig. 6.9 for further information

- View Solids

  This will show all objects as solid, can also be set to View Solids/Lines, View Lines (wireframe) or View Points

- Clip

  This will open up a drop down menu to clip the simulation view, with a chosen normal direction and coordinate to place the clip at.

- Toggle Axes

  A toggle button to show or hide the axes.

- Show Scale

  This will allow changing the scale shown be default, from nanometers to kilometers. Note that all values will remain in meters or as defined by the user, this is just used to rapidly change the scale shown in the setup menu.

- Perspective View

  This will reset to a perspective view for the 3D view.

- Axis Drop-down Menu

  Drop-down menu for choosing the axis from which the object is viewed.

- Reset Position

  Pressing this button will reset the camera view to be along the axis selected in the drop-down menu.

6.17.2 Navigating

Navigation in 3D space is possible under keyboard and mouse control.

- Rotate

  Holding down the right mouse button and dragging it will rotate the camera position around the object being viewed.

- Pan

  Holding down the left mouse button and the Shift key pans the view in a plane without rotating the viewed object.

- Zoom

  Using the mouse wheel, the view can be zoomed in or out. If the system does not have a mouse wheel, then holding down the left mouse button and the Control key and moving the mouse up and down will also zoom in and out.

6.17.3 Database View

The *Database View* is for viewing and adding materials to your simulation. Initially, the *Database View* is blank, but upon importing a materials (.vmat) file, the view is populated with a table of data from the file. See Fig. 6.23.

You can switch between files, if more than one file is open, by changing the left drop-down menu. You can remove a file by first switching to the file you would like to close, and then clicking on the *Remove File* button.

You can add materials to your simulation by highlighting the particular material you are interested in, and then clicking on the *Add to Simulation* button.

For more information on materials, please see *Materials*.
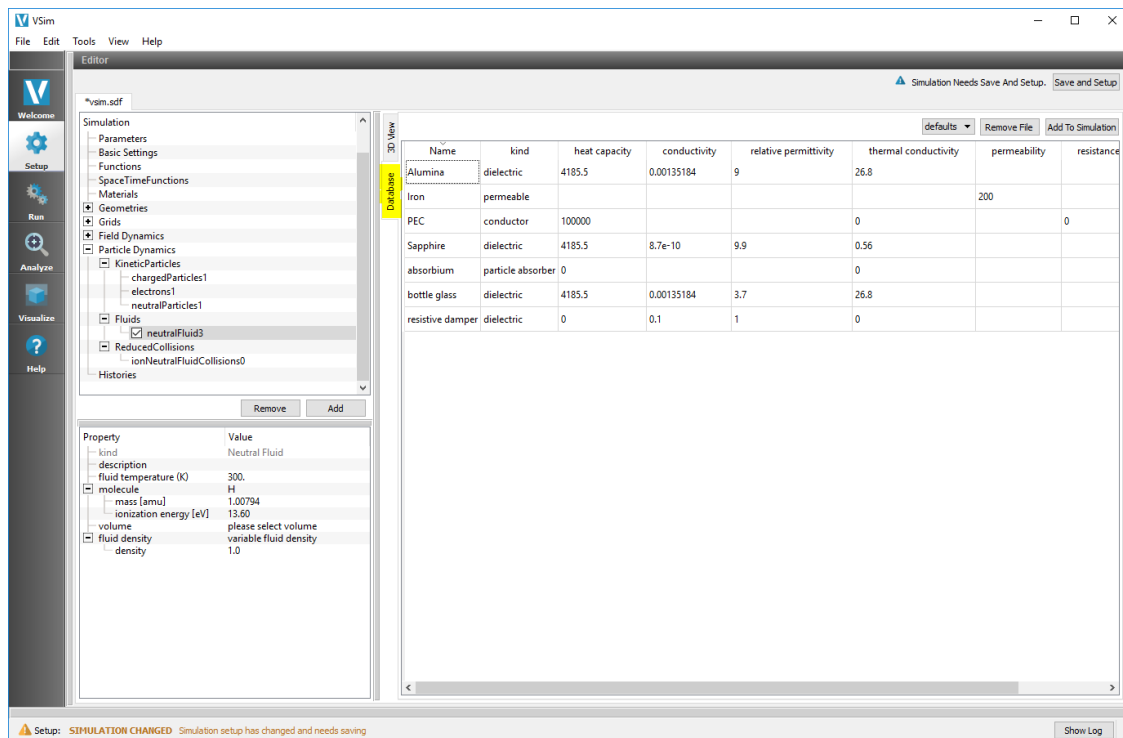
Fig. 6.23: The Database view

TEXT SETUP

7.1 Introduction to Text Setup

VSim offers a text-based setup for simulation in addition to its visual setup method. Rather than constructing a simulation through a graphical interface, the text setup allows the user to directly access the text-based (.pre) input file instead. Though this method requires the user to have a more thorough knowledge of the simulation grid, field properties, and other general characteristics governing the simulation, it also allows the user to combine fields and particles in unique ways. It also allow one to use some features that are currently not available through visual setup.

7.2 Setup Basics

7.2.1 Setup Window for Text-Setup Simulations

You can open a simulation as described in *Opening an Existing Simulation*. After you open a simulation, VSim-Composer displays the **Setup** window that contains an *Editor* pane with some easy-to-edit parameters and a short description and image of the simulation. This is shown in Fig. 7.1.

From here you can modify the exposed parameters of the simulation, in order to explore the dynamics for different values.

Pressing the *View Input File* button, boxed in red at the top, takes one to the *Input File View*, shown in Fig. 7.2.

This gives full access to the text-setup input file, so that you can add parameters, modify expressions, even modify the algorithms used in the simulation.

Additionally, one can see all the files in the simulation directory by pulling the separator bar to the right, as shown in Fig. 7.3.

The following sections will go through each of the components of the **Setup** window. We will then delve into the more in-depth .pre file, including general properties, elements, and recommended structure. See Fig. 7.4.

Navigation Pane and Simulation Files

The *Navigation* pane contains a list of *Simulation Files*, and can be accessed by clicking and dragging on the vertical bar that separates the *Icon* panel and the *Editor* pane.

To enable convenient viewing of the list of simulation files, VSimComposer allows you to specify in what order as well as which type(s) of files you would like to view. *Smart Grouping* causes similar types of files to be displayed in the same area of the *Simulation Files* tab list. Turning off *Smart Grouping* causes files to be displayed in alphabetical order rather than by type. *All Files* indicates that you want to see all available files involved in the simulation. You could choose to limit your view to only *Simulation Files*, which are files such as input files and macros that can be
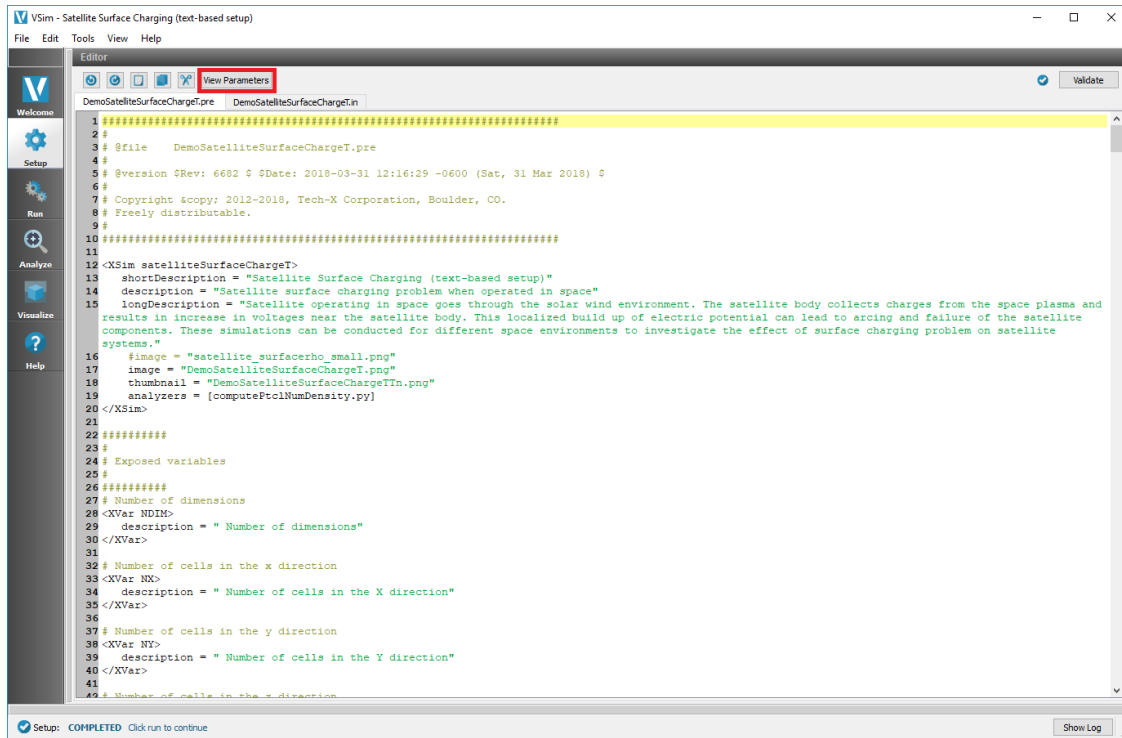
Fig. 7.1: Parameters view of the text-setup setup window.
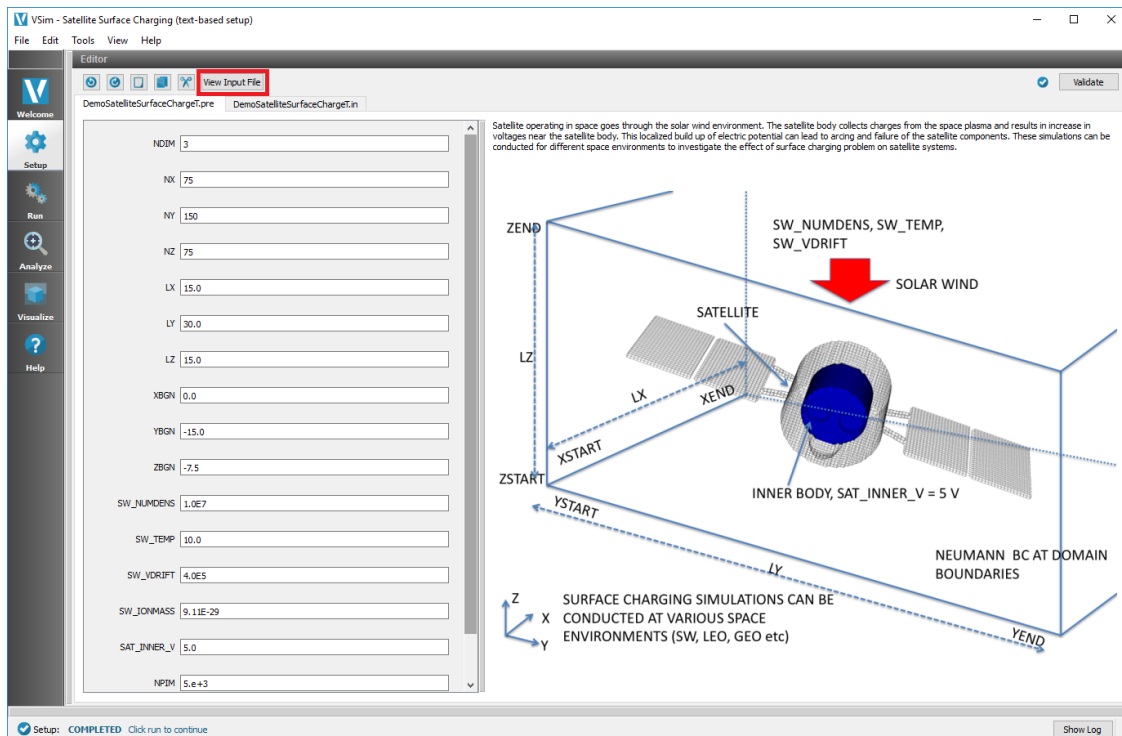
Fig. 7.2: Input file view of the text-setup setup window.
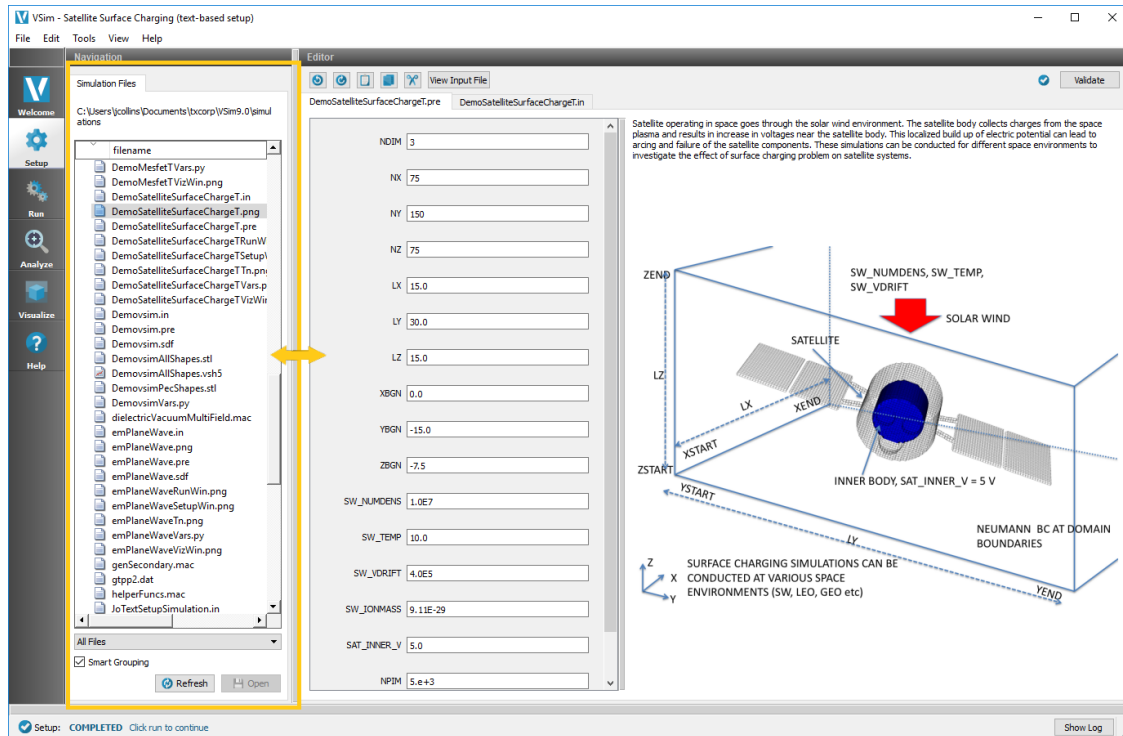
Fig. 7.3: Setup Window Files View
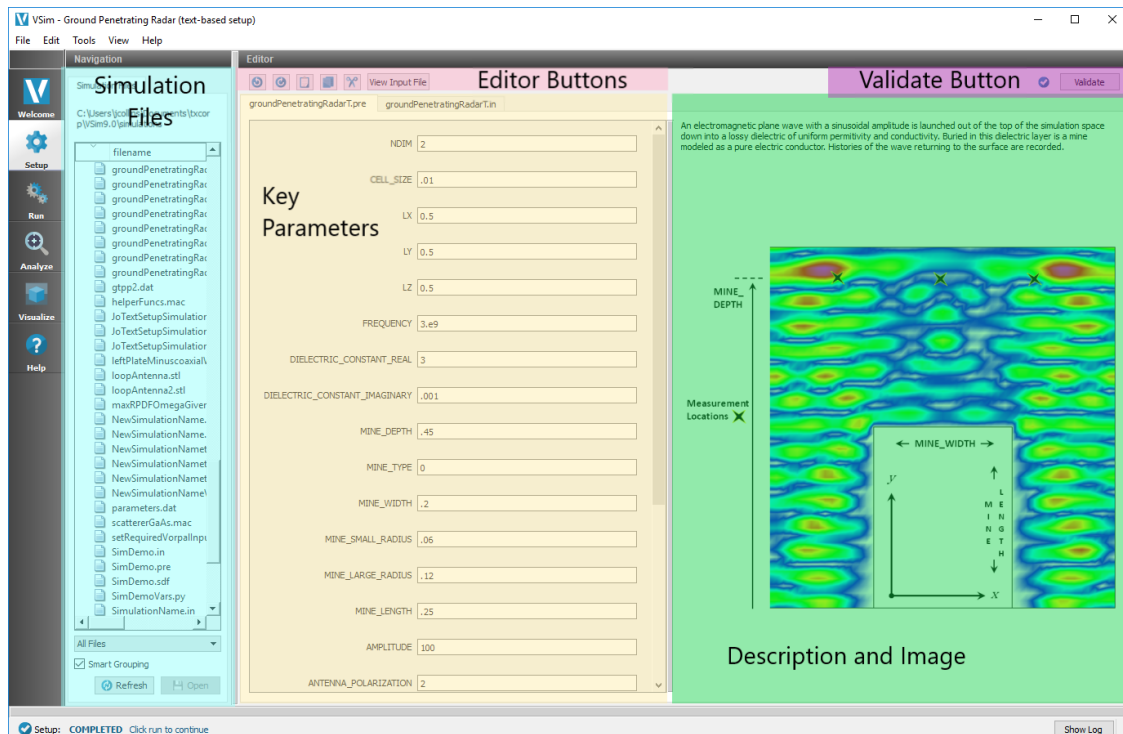
Fig. 7.4: The main parts of the **Setup** window

edited in the VSimComposer Editor pane, or *Text* files, which include all types of human-readable file formats, or *Data* files, which include incremental dump files and output files that can be visualized.

Key Parameters

All the example files in VSimComposer come with key parameters, allowing the user to easily adjust basic parameters of the simulation.

By holding the mouse over the key input parameter name, a description of what exactly the variable does will pop up. Many examples can be significantly modified with just the key input parameters to provide a good starting point for a different application. See Fig. 7.4.

If you have opened one of your own simulations that is not based off of an example, you may not have these key parameters. For more information on how to create these in your own file, please see *Key Parameters*.

Editor Buttons

From left to right, the editor buttons are:

- Undo
- Redo
- Paste
- Copy
- Cut
- View Input File / View Parameters

These buttons can be used for editing the key parameters as well as editing the .pre (or text-based) file.

If you would like to see the .pre (text-based) file, simply click on the *View Input File* button. This will bring you to the traditional .pre file. Modifications made in the Key Parameters window will carry over if you switch to input file view.

Validate Button

If many modifications to the input file have been made, it is suggested to first validate the simulation. After the *Validate* button has been clicked, the input file will be checked to make sure that there are no errors.

Description and Image

The far right of the **Setup** window holds a description of the simulation and a corresponding image.

If you have opened one of your own simulations that is not based off of an example, you may not have the description and image. For more information on how to create these in your own file, please see the discussion on the XSim Block in *Key Parameters*.

Hidden Log View and Find/Replace

Note: A *LOG VIEW* can be shown by clicking and dragging on the horizontal bar at the bottom of the Composer window.

Note: *Find/Replace* and *Results* tabs can be shown by clicking and dragging on the second and smaller horizontal bar above the bottom of the Composer window.

VSimComposer notifies you of the actions that it is taking in the *LOG VIEW*. This includes output from the validation of the input file, and imported files, among other things.

When editing the .pre file, VSimComposer has a *Find/Replace* tab to help naviagate the text. You can access this by going to *Edit –> Find and Replace* or using the shortcut keys Ctrl+F.
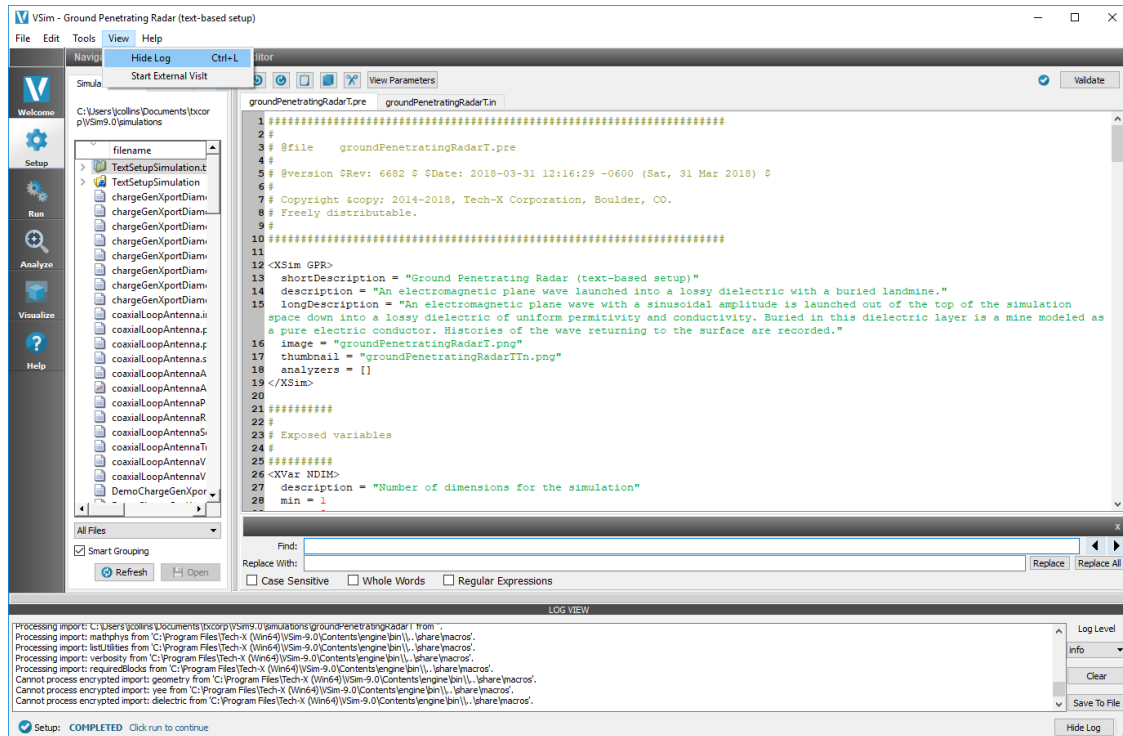
Fig. 7.5: **Setup** window tab for output message

7.3 Text-based (.pre) Input File Structure

7.3.1 Input .pre Files

We will now get into the actual construction of a text-setup simulation. Start by either going to *File –> New –> Text-setup Simulation*, or by opening any text-based simulation with the intention of editing its .pre file. Once your simulation is open, click on the *View Parameters* button at the top of the **Setup** window.

7.3.2 .pre File Structure

We will begin by discussing general .pre file syntax and concepts, including:

- Comments
- Variables

- Globals
- Blocks
    - Top-level Blocks
    - Nested Blocks

We will then move into properties that you will want to be able to implement and alter to fit your specific simulation needs:

- Geometries
- Grids
    - Decomposition
- Fields
    - Field Boundary Conditions
    - EmField and MultiField
- Particles
    - Particle Sources
    - Particle Sinks
- Fluids
- Macros
- XSim and XVar Concepts

The concepts and properties discussed in this section are only introductory. It may be worthwhile to open *Text-based Setup* examples and explore their .pre files in order to more clearly see the above concepts in action.

More information on specific types, parameters,and other options can be found in the *VSim Reference Manual*. For details on further simulation customization, including information on importing your own analyzers and macros, can also be found in *VSim Customization*.

Comments

Comments are helpful in either describing various parts and functions within your code, and can also be used to visually break up your code into different sections. The loosely defined "sections" that will be used to describe .pre file structure are technically commented sections.

Comments can be entered through one of two ways:

- Type in a pound sign (#), and then either start your comment on the same line or a new line.
- Use the opening and closing tags <Comment>, </Comment> before and after your comment text.

Note: Tech-X recommends that you always update your comments when you make changes to an input file. The reasoning behind a change may become unclear if you do not provide comments that explain why you made the change. Input files with old, out-of-date comments are difficult to work with.

7.3.3 Variables

User-Defined Variables

Aside from the global variables built into VSim (see *Globals*), you can also establish your own variables. They are defined through assignment, similar to many other programming languages. The syntax for defining a variable with an expression is:

```
$ VARIABLE = EXPRESSION
```

where VARIABLE is the name of your variable and EXPRESSION represents any valid expression. (See *Expression Evaluation* for more details.)

Note: Each line defining a variable must begin with a dollar sign (`$`).

VSim's Python preprocessor will not try to substitute a variable on the left hand side of an equal sign (=). For example, the following code snippet:

```
$ charge = 1.6e-19
charge = charge
```

results in

```
charge = 1.6e-19
```

You can also use existing variables to define your variables. For example, you can use the length and number of cells along the x direction of your simulation in your definition of `DX`, as shown below:

```
$ DX = LX/NX
```

You still must place a dollar sign (`$`) in front of the variable you want to define.

Scoping and Evaluation

Variables in VSim are *scoped*. This means that the effect of a variable's definition is confined to the macro or block in which that variable is defined. Whenever VSim enters a macro or a new input block, it enters a new scope.

In the case in which a variable is defined in multiple scopes, Vorpal ignores the previously-defined variable for the duration of the current scope. If the variable is defined more than once in the current scope, the new value overrides the previous value defined in the current scope.

A scope is closed once VSim leaves the block or macro. That is, the variable's definition no longer has an effect once VSim has used the variable's value in the macro or block where it is defined and then proceeded to a different block or macro. Scoping allows the next block to be free to redefine the value of the variable for its own purposes.

Mathematical Expressions

You can put mathematical expressions directly in an input file's blocks by encapsulating them between dollar signs (`$ math $`). When you validate an input file, the expressions within the dollar signs are evaluated.

For example, in the `esPtclInCell.pre` file, there exists a variable that is defined as:

```
$ NX1 = NX + 1
```

and it appears in a vector describing the upper bounds of the simulation, `[NX1 0 0]`. However, you can also just use a mathematical expression directly in the bound definition, so you'd have:

```
[$NX + 1$ 0 0]
```

instead of the separate lines defining the variable and the upper bound.

Expression Evaluation

VSim evaluates expressions by interpreting them as Python expressions, which are composed of tokens. A token is a single element of an expression, such as a constant, identifier, or operation. The preprocessor breaks the expression into individual tokens, then performs recursive substitution on each token. Once a token is no longer substitutable, the preprocessor tries to evaluate it as a Python expression. The result of this evaluation will then be used as the value of this token. All the token values are subsequently concatenated and again evaluated as a Python expression. This result will then be assigned to the symbol.

Tokenizing, the act of breaking a string into tokens, is performed according to the lexical rules of Python. This means that white spaces are used to delimit tokens, but are otherwise entirely ignored.

Note: A string within matching quotes is treated as a single token with the matching quotes removed.

The processed input files generated by VSim are sensitive to white spaces; as a result, VSim has to re-introduce white spaces in the translation process. By default, tokens are joined without any white spaces. However, if both tokens are of type string, then a white space is introduced. Also, tokens inside an array (delineated by `[ ]`) are delimited by a white space.

See the Python documentation on the official Python website (http://www.python.org) for more information about Python expression.

Parameters

Parameters can be integers, floating-point numbers, or text strings. The format of the parameter value determines the type of parameter. For example:

- `x = 10` indicates an integer
- `x = 10.0` indicates a floating-point integer
- `x = ten` indicates a text string

Some parameters accept any text string (within reason). Other parameters accept only a choice of text strings.

Use a decimal point to specify a floating point number. You must write floating-point numbers with a decimal point so that they will not be interpreted as integers. If you want to assign a floating-point value to an integer parameter, make sure you write it as `3.` (with a decimal point) rather than only the numeral `3` (without a decimal point). If you write the number as an integer, VSim will interpret it as such. This will likely produce unexpected results.

If VSim can parse a value, such as `42`, as an integer, it will do so. If VSim cannot parse the value as an integer, it will attempt to parse it as a floating-point number. If VSim cannot parse the value as either an integer or floating-point number, it will parse it as a string of text.

Check that you have correctly defined parameter values. If you incorrectly define a parameter that has a default value, the default value will be used and potentially produce unsatisfactory results. If you incorrectly define a parameter that does not have a default value, the computational engine may crash, fail to compute the physics of the simulation, or ignore the incorrectly defined parameter and give you unsatisfactory results.

Do not specify a parameter twice. If you do, the second occurrence of the parameter in the processed .in file (produced from the .pre file) will be used. Although parameters and input blocks can be defined in many different sequences, if you follow the recommendations in this guide, you should not have a problem with specifying parameters twice.

Vectors of Parameters

Vectors of parameters are enclosed by brackets `[ ]` with white space used as separators. For example:

- `x = [10 10 10]` indicates a vector of integers
- `x = [10. 10. 10]` indicates a vector of floats

7.3.4 Globals

Global variables can be declared outside of any particular block, and they control various general aspects of the simulation. For example, the global variable `dimension` defines the number of dimensions for your simulation. When running your simulation from the command line, you can override values set in your .pre file for required global variables by using the variable's command line parameter to define a new value. Not all global variables have command line parameters, in which case they must be defined in your .pre file.

Required global variables are as follows:

floattype (**string, required**) Variable that defines the precision of real numbers in the simulation. For greater precision, use the value `double`, otherwise use the option `float`. You must define `floattype` in your input file.

Example use of `floattype`:

```
floattype = double
```

dimension (**integer, required**) Variable that defines the dimensionality (1D, 2D, or 3D) of the simulation. Its command line parameter is `-dim`.

Example use of `dimension`:

```
dimension = 2
```

dt (**real, required**) Time step size for your simulation. When choosing the step size, you also must consider stability requirements. For example, you must satisfy the Courant condition when selecting a step size in electromagnetic simulations. Its command line parameter is `-dt`.

Example use of `dt`:

```
dt = 1.49731212265e-16
```

nsteps (**integer, required**) Number of steps to take. (In the case of a restart, `nsteps` is the number of additional steps.) Its command line parameter is `-n`.

Example use of `nsteps`:

```
nsteps = 22277
```

Optional global variables include the following, and are discussed in VSim Reference: Global Variables.

- **dumpPeriodicity**
- **dumpSteps**
- **maxcellxing**

- **sortPtcls**

- **dnSortMin**

- **dnSortMax**

- **useGridBndryRestore**

- **copyHistoryAtEachDump**

- **stepPrintPeriodicity**

- **timingAnalysisPeriodicity**

Global Variables Specific to Moving Windows

The moving window feature allows the simulation window to move at the speed of light in the chosen direction. This feature is used to reduce the size of the simulation box while following the physics phenomenon of interest, such as a laser pulse or a particle beam that is propagating at a velocity close to the speed of light.

This feature and its parameters are as follows:

- **moving window**

- downShiftDir

- downShiftPos

- OAFuncshiftSpeed

These, as well as example code, are given in detail in VSim Reference: Global Variables.

Global Variables from Inside a Block

It is possible to declare your own global variable in VSim. This is done by first defining the variable, then declaring it global. For example:

```
<Block>
  $ X = 4
  $ global X
</Block
```

This will cause the variable X to equal 4 outside of the block. The variable must be defined and declared global on separate lines. For example,

```
$ global X = 4
```

will not define X as a global variable with value 4.

Defining Basic Simulation Parameters

Your simulation will include at least the following global parameters:

- dimension

- floattype

- dt

- nsteps

- dumpPeriodicity

You can either set these equal to actual values, or you can set them equal to variables already defined in your .pre file.

7.3.5 Blocks

Input blocks are used to create simulation objects. The block is enclosed by opening and closing tags, such as:

```
<Grid globalGrid>
.
.
.
</Grid>
```

The tag determines the following:

> **Object type**: Begins with a capital letter and is in "camelCase". For example, `Grid` or `EmField`.

> **Object name**: Indicated with a lower case letter and is in "camelCase". For example, `globalGrid`.

You use the object name to refer to the object in other input blocks. For example, in the input block for a particle object, you may refer to the name of the electromagnetic field object.

Input blocks can be nested. For example, input blocks for boundary conditions are nested within the input block for an electromagnetic field.

Implementation Kind

Most Vorpal blocks, including both top level and nested blocks, have several algorithms from which you may choose by specifying a parameter named `kind`. For example, an EMField may be modeled as a Yee field, a direct sum field, a constant field, or any of several other types. You use the `kind` parameter to select a particular implementation. Each block description in this manual lists the available `kind` parameter settings for that block.

Note: If you don't see the implementation kind you need for a given field block, you may wish to consider learning to use the `MultiField` blocks to define your own implementation. In addition, Tech-X Professional Services is available on a contractual basis to create custom implementations and simulations. Contact Tech-X at sales@txcorp.com to discuss consulting options.

Example blocks specifying `kinds`:

```
<EmField myExternalField>
  kind = funcEmField
  <STFunc E0>
    kind = expression
    expression = EX_1 * cos(K_PE * x) * H(DRIVE_TIME - t)
  </STFunc>
</EmField>
```

Top Level Blocks

Blocks that appear at the top level of the input block hierarchy define the basic characteristics for the simulation as a whole and for other blocks that will be used in the simulation. Some top level blocks, such as **Grid**, **Decomp**, **SumRhoJ**, can appear only once in an input file, and are denoted as singletons. Every simulation must, without exception, contain a **Grid** and **Decomp** block.

Other blocks, such as **Species**, can be used as many times as needed in the input file. You will find detailed descriptions of blocks in the *Text Setup* section of VSim Reference.

Top level blocks include:

Grid (singleton): Determines the simulation size and relationship of physical coordinates to cell indices. Grid is required in every input file block.

Decomp (singleton): Determines the domain decomposition and periodicity. Decomp is required in every input file block.

SumRhoJ (singleton): Defines the properties of the charge and current density 4-vector field.

GridBoundary: Defines any embedded boundaries.

EmField: Defines any electromagnetic fields.

ComboEmField: Defines any combinations of electromagnetic fields.

Fluid: Defines any fluids.

Species: Defines any particles.

MonteCarloInteractions: Defines any random processes that may occur between different objects in the simulation, such as collisions or ionization processes.

MultiField: Defines general field blocks whose parameter and variable values may be adjusted during the simulation.

ScalarDepositor: Alternate method to deposit charge from charged particles in a simulation into a depField.

VectorDepositor: Alternate method to deposit current from charged particles in a simulation into a depField.

History: Used to record data from a simulation over time.

For more details, please refer to the respective sections in *VSim Reference*.

Nested Blocks

While top level blocks are used at the top of the input file hierarchy, nested blocks are included within other code blocks. For example, a Vorpal **Species** block can contain a **ParticleSource** block that describes how that species is inserted into the simulation. The **ParticleSource** code block is, therefore, said to be nested within the species block. Nested blocks are noted in the descriptions of those blocks that can contain them.

A nested block applies only to the block that contains it. For example, you can use a **BoundaryCondition** block to affect an **EmField**. You could then specify different boundary conditions for a second **EmField** block.

Particle species can also contain other objects. For example, you can use **ParticleSource** and **ParticleSink** blocks in **Species** to describe where particles are to be placed into and removed from the simulation. By using these blocks' kind parameters, you describe how the emission or absorption is to be accomplished. You are not limited to defining blocks using a single level of nesting. The **ParticleSources** contained inside a particle species like <Species electrons> also contain a Vorpal **STFunc** block. Taken all together, these blocks denote the space-time function used to describe particle emission.

Example of nested blocks:

```
<Species electrons>
  kind = relBoris
  charge = -1.6e-19
  mass = 9.109e-31
```

```
emField = myEmField
# Nominal density and particles per cell at that density
nominalDensity = 4.41204859999e+22
nomPtclsPerCell = 2.
# Particles loaded in a ramp
<ParticleSource stepSrc1>
  kind = bitRevDensSrc
  density = 4.41204859999e+22
  lowerBounds = [2.5e-07 -2.5e-05 -2.5e-05]
  upperBounds = [5e-06 2.5e-05 2.5e-05]
  doShiftLoad = 1
  vbar = [0. 0. 0.]
  vsig = [0. 0. 0.]
  # Unit probability
  <STFunc macroDensFunc>
    kind = constantFunc
    amplitude = 1.
  </STFunc>
</ParticleSource>
<ParticleSource stepSrc2>
kind = bitRevDensSrc
density = 4.41204859999e+22
lowerBounds = [5e-07 -2.5e-05 -2.5e-05]
upperBounds = [5e-06 2.5e-05 2.5e-05]
doShiftLoad = 1
vbar = [0. 0. 0.]
vsig = [0. 0. 0.]
# Unit probability
<STFunc macroDensFunc>
  kind = constantFunc
  amplitude = 1
</STFunc>
</ParticleSource>
# Particles out left are removed
<ParticleSink leftAbsorber>
  kind = absorber
  minDim = 1
  lowerBounds = [-1 -1 -1]
  upperBounds = [0 21 21]
</ParticleSink>
# Particles out right are removed
<ParticleSink rightAbsorber>
  kind = absorber
  minDim = 1
  lowerBounds = [40 -1 -1]
  upperBounds = [41 21 21]
</ParticleSink>
</Species>
```

Notice that adequate comments are provided to explain what is going on in each nested block.

Note: Tech-X recommends that when you nest input blocks, use an appropriate amount of indentation to improve the readability of the input file.

7.3.6 Geometries

You can include geometries in text-based simulations, either by importing existing geometries or creating them in your prefile.

Creating Geometries

One can create primitives and perform operations on them, just as in the visual setup. By importing the `geometry` macro as detailed above, one gets a number of predefined primitives, as well as functions that can be used to perform operations on these primitives, or to create custom shapes.

Along with the creation of primitives and/or functionally-defined shapes, the following operations will be detailed herein:

- Moving and rotating shapes
- Advanced filling and voiding
- Making your own primitives

Primitives

The `geometry` macro includes a number of primitives, including some that are available in visual setup as well as ones that are unique to the macro. Primitives that are also found in visual setup are as follows:

- `geoBoxP`
- `geoCylinderXP`
- `geoPipeXP`
- `geoConeXP`
- `geoSphereXP`
- `geoTorusX`

The primitives that are unique to the macro are below.

- `geoQuadrilateralSlabXP`
- `geoHemiSphereXP`
- `geoBiParabolicSlabXP`
- `geoTriangleSlabXP`
- `geoRndCylinderXP`
- `geoRndRectangleSlabXP`
- `geoEllipsoid`
- `geoHemiEllipsoidXP`
- `geoEllipticalCylinderXP`
- `geoEllipticalConeXP`

To use these primitive functions, one must assign a geometry name to appear on the first in the parentheses, followed by the primitive function and values you've entered for the parameters. Below is an example showing the creation of a hollow cylinder with the `geoCylinderXP` function, using the described syntax.

```
fillGeoExpression(hollowCylinder, geoPipeXP(x,y,z,INNER_RADIUS,RADIUS,LXO))
```

The parameters used above are as follows, though one could also just input numerical values rather than parameters or constants:

- x

- y

- z

- **INNER_RADIUS** Inner radius of the cylinder.

- **RADIUS** Outer radius of the cylinder.

- **LXO** Length of cylinder in the axial direction (x).

The parameters required for each of the primitive functions in the macro are detailed fully in the section on the macro file in VSim Reference.

Moving and Rotating Shapes

The above shapes are defined around points of symmetry; cylinders and spheres are centered around the origin, and rectilinear shapes begin at the origin and extend out in the x-direction.

However, one can provide offsets in any direction–x, y, or z–by subtracting the desired offset from the respective variable parameter. For instance,

```
fillGeoExpression(hollowCylinder, geoPipeXP(x-0.03,y+0.04,z,INNER_RADIUS,RADIUS,.1))
```

would result in our cylinder being offset from the origin by 3 centimeters in the x-direction and 4 centimeters in the y-direction. Its length in the z-direction would be 10 centimeters.

One can also rotate the shape so that it is centered about a different axis. For example, if the cylinder needed to be centered about the z-axis, the .pre file input would be:

```
fillGeoExpression(hollowCylinder, geoPipe(z,x,y,INNER_RADIUS,RADIUS,.1))
```

Notice that the x, y, and z entries from before have changed so that they now are z, x, and y. The first entry is the axial direction.

You can mimic the Boolean operations that are available with visual setup by using the

```
voidGeoExpression fillGeoExpression
```

functions with the primitives you have already defined in your simulation. The first example below walks through joining two primitives, one being a box named `box1` and the other being our `hollowCylinder`.

```
fillGeoExpression(box1, hollowCylinder)
```

Advanced Filling and Voiding

There are two main methods of creating custom shapes, function-defined and primitive-based.

The function-based way of creating shapes involves first creating functions that describe your desired shapes, which can be done through implementation of Heaviside functions. You can then use these functions as input for either the `fillGeoExpression` or `voidGeoExpression` functions. We can use this process as an alternate means of building our hollow cylinder from before, beginning with the function definition:

```
<function cylinderFunc>
  H(-INNER_RADIUS^2 + x^2 + y^2)*H(RADIUS^2-x^2-y^2)
</function>
```

where the INNER_RADIUS and RADIUS parameters are the same as those used in the hollowCylinder primitive above.

The geometry itself can then be defined by simply determining a name as the first parameter–here, we have chosen *hollowCylinder* again–and then invoking the function defined above.

```
fillGeoExpression(hollowCylinder, cylinderFunc(x,y,z))
```

The `voidGeoExpression` function, on the other hand, designates a certain volume of space to be removed from another. For example, this can be applied to a box, which can then be "subtracted" from a larger box. These operations are implemented in a macro, whose name is that of the overall geometry.

```
<macro boxes>
  voidGeoExpression(smallBox, geoBoxP(x,y,z,0.05,0.05,0.05))
  fillGeoExpression(bigBox, geoBoxP(x,y,z,0.1,0.1,0.1))
</macro>
```

Making Your Own Primitives

If you want to make your own primitive that can then be used repeatedly in your simulation, you can define your geometry in the macro format and then call the macro from multiple places in your .pre file. For example, we can define our hollowCylinder as below:

```
<macro hollowCylinder>
  H(-INNER_RADIUS^2 + x^2 + y^2)*H(RADIUS^2-x^2-y^2)
</macro>
```

Importing Geometries

One can import both .stl and Python-defined shapes. CAD geometries can be used in conjunction with CSG primitives built in VSim.

CAD (.stl) Shapes

You can import existing shapes whose files end with the .stl extension by using the (slightly different) functions `fillGeoCad` and `voidGeoCad`. There also exists the optional function `fillGeoFastCAD`.

These functions' parameters are detailed in full in the `geometry` macro file section, but a basic example of `fillGeoCad` being used to import an .stl file is given below:

```
fillGeoCad(hollowCylinder, hollowCylinder.stl, False, 1.0,[0 0 0])
```

where "hollowCylinder" is the name of the object in VSim and "hollowCylinder.stl" is the .stl file being imported. The last three entries, "False," "1.0," and "[0 0 0]," are default values, and as such the hollowCylinder object has imported with 1-1 scaling and no translations.

Importing a Python File

To import a Python-defined geometry, one can use the functions `fillGeoPython` and `voidGeoPython` in conjunction with a Python function that is defined in a .py file.

```
fillGeoPython(hollowCylinder, makeHollowCylinder)
```

The Python file must have the same name as the input file name and contain the function you wish you use. The function should be equal to 1 inside the geometry object and to 0 outside.

7.3.7 Grid

Define the Grid

You can use the `Grid` block to define the simulation grid. You can either use variables or the desired values directly for parameters such as numCells, lengths, etc.

```
<Grid globalGrid>
  kind = uniCartGrid
  numCells = [NX NY NZ]
  lengths = [LX LY LZ]
  startPositions = [XBGN YBGN ZBGN]
</Grid>
```

A full description of all grid types and parameters can be found in the Grid section of VSim Reference.

Defining the Grid in Different Coordinates

You can set the simulation grid to be in cylindrical coordinates simply by setting

```
coordinateSystem = Cylindrical
```

Expressions will switch from relying on x and y coordinates to z and r automatically. You should still use x and y in the actual expressions.

More information can be found in the Cylindrical Coordinates section of VSim Reference.

Define the Decomposition

You can use the `decomp` block to define the decomposition to be used in the simulation, particularly if you want something other than Vorpal's default decomposition method. It is also in this block that you can ensure periodic boundary conditions are incorporated in your simulation.

```
<Decomp decomp>
  kind = regular
  periodicDirs = [1 2]
</Decomp>
```

More details on decomposition and related parameters can be found in the Decomp section of VSim Reference.

7.3.8 Fields

A number of fields can be implemented in your simulation, including different types of electromagnetic and electro-static fields. Thus, there also exist a variety of ways to incorporate fields in VSim. In this section, we will cover general concepts surrounding fields, including:

- Boundary conditions
    - Symmetry boundaries
    - Fully periodic systems
    - Modeling a time-dependent or space-dependent value on a boundary
- Signal creation
    - Creating a circularly polarized pulse
    - Launching a wave from the x-upper boundary of a simulation

We will then look at an example of field blocks in action. We will first look at an electrostatic field, which is created through an `EmField` block. We will then look at an electromagnetic field that uses the `MultiField` block. This block in particular allows the user to control more parameters for their fields, make changes pertaining to processing efficiency, or even develop and apply their own algorithms. This object incorporates three general sub-objects:

- **Field** Specification of data.
- **FieldUpdater** Update operations to be performed on the the data.
- **UpdateStep** Algorithmic sequencing of the update operations.

Boundary Conditions

You can implement Dirichlet, Neumann, MAL, and other boundary conditions for your simulation, just like in visual setup.

We will provide a few examples of common boundary conditions you may want to incorporate in your simulation, including:

- Symmetry boundaries
- Fully periodic systems
- Time-dependent or space-dependent values on a boundary
- Launching a wave from the x-upper boundary

Symmetry Boundaries

A Neumann boundary can act as a symmetry boundary. For instance, placing Neumann boundaries appropriately on three faces of a cube will turn the simulation into a symmetric simulation of an 1/8th of a larger cube.

Fully Periodic Systems

You can model a system that is periodic in all directions–that is, one that has no boundaries. In a fully periodic system, however, the overall charge must be zero, as demonstrated in this equation:

$$\int_x^{x+L} \frac{\partial E}{\partial x} dx = E(x+L) - E(x) = \frac{1}{\epsilon_0} \int_x^{x+L} \rho dx = \frac{L}{\epsilon_0} \langle \rho \rangle \tag{7.1}$$

where $\langle\rho\rangle$ is the average charge density over the integration interval. Zero net charge can be enforced by specifying the electrostatic solver parameter `enforceZeroNetCharge = true`.

For more information, see [BL04].

Modeling a Time-Dependent or Space-Dependent Value on a Boundary

An electromagnetic boundary condition may use time signals or spatial profiles. For example, the `emPlaneWave.pre` file's `xLowerWaveLauncher` boundary condition input block uses a spatial profile, and so `kind = variable`.

The `function` parameter in a boundary condition input block also may be an expression, and the expression may be any standard function of space and time.

The `emPlaneWave.pre` file uses the `kind = expression` technique.

```
<BoundaryCondition xLowerWaveLauncher>
  # Value given by function
  kind = variable
  lowerBound = [0 0 0] # Lower limits
  upperBound = [1 $NY+1$ $NZ+1$] # Upper limits
  components = [1] # Ey polarized
  <STFunc component1>
    kind = expression
    expression = 1.e6*sin(OMEGA*t)
  </STFunc>
</BoundaryCondition>
```

For the complete content of the `emPlaneWave.pre` file, see the example Electromagnetic Plane Wave in VSimBase.

Signal Creation Examples

Creating a Circularly Polarized Pulse

The steps to create a circularly polarized pulse are:

1. In the electromagnetic boundary condition input block, change the components parameter vector to launch both a y and z component: `components = [1 2]`.

2. Give the amplitudes parameter vector a value for each component: `amplitudes = [AMPLITUDE AMPLITUDE]`.

3. Give the phases parameter a value for each component. To make a circularly polarized pulse, the phases for the two components should be $\pi/2$ radians (90 degrees) apart: `phases = [0. 1.57]`.

The `widths` parameter controls the spatial extent of the wave.

Launching a Wave from X-Upper Boundary of Simulation

The steps to change a wave launcher from the x-lower to the x-upper boundary follow:

1. Change the lower and upper bounds of the wave launcher boundary condition to the x-upper end of the grid.

Note: The lower bounds of a boundary condition at the x-upper end of the grid are in the physical domain.

7.3. Text-based (.pre) Input File Structure | **91**

2. Adjust the x-upper conducting boundary to zero out only the y component.

3. Adjust the x-lower conducting boundary to zero out both y and z components. Be sure to change both the amplitudes and components vector.

The simulation occurs in the lab frame. This means the pulse will strike the x-lower boundary and reflect back after roughly fifty steps for the given time step and simulation size. To avoid reflections, you need to include an absorbing boundary at the x-lower end.

Defining the EM Field and Boundary Conditions in VSim

Now that we've discussed boundary conditions, pre-conditioners, and solvers, we are ready to try adding fields to simulations.

You can implement multiple electric and magnetic fields in your simulation. Here, we will show examples of both the `EmField` and `MultiField` objects being used.

Electrostatic Field Example

We will begin with `EmField`, where we will define an electrostatic field covering the entire grid.

```
<EmField yseField>
  kind = yeeStaticEmField

  # Setting potential V = 0 on left wall of simulation
  <BoundaryCondition left>
    kind = dirichlet
    minDim = 1
    lowerBounds = [0 0 0]
    upperBounds = [1 NY1 NZ1]
    # NY1 = NY + 1, NZ1 = NZ + 1 (these are where the guard cells are)
    <STFunc voltFunc>
      kind = constantFunc
      amplitude = 0.
    </STFunc>
  </BoundaryCondition>

  # Declaring our solver and preconditioner
  <Solver yseSolver>
    kind = gmres
    precond = multigrid
    smoother = GaussSiedel
    nLevels = 12
    threshold = 0.08
    tolerance = 1e-8
  </Solver>

</EmField>
```

Electromagnetic Field Example

The `MultiField` block, in comparison, does need a number of nested blocks that describe the fields involved, as well as their boundary and initial conditions, field updaters, and update steps. Below, we have written up an example with an electric and magnetic field.

```
<Multifield myFields>
 # The default kind for MultiField is Null, which is what we will use here.

  <Field E>
    kind = regular
    numComponents = 3
    labels = [E_x E_y E_z]

    # Set E_y = 1.0 on y-lower boundary initial condition
    <InitialCondition eInitial>
      kind = constant
      lowerBounds = [-1 0 -1]
      upperBounds = [NX1 1 NZ1] # NY1 = NY + 1, NZ1 = NZ + 1 (guard cells)
      indices = [1]
      amplitudes = [1.0]
    </InitialCondition>

    # Set E_x = 0.0 (conductor) on x-lower boundary
    <BoundaryCondition eBound>
      kind = constant
      lowerBounds = [0 -1 -1]
      upperBounds = [1 NY1 NZ1]
      indices = [0]
      amplitudes = [0.]
    </BoundaryCondition>

  </Field>

  # Define simple magnetic field
  <Field B>
    kind = regular
    numComponents = 3
    labels = [B_x B_y B_z]
  </Field>

  <FieldUpdater yeeAmpere>
    kind = yeeAmpereUpdater
    readFields = [B]
    writeFields = [E]
  </FieldUpdater>

  <FieldUpdater yeeFaraday>
    kind = yeeFaradayUpdater
    readFields = [E]
    writeFields = [B]
  </FieldUpdater>

  <UpdateStep firstStep>
    toDtFrac = 1.0
    updaters = [yeeAmpere]
  </UpdateStep>

  <UpdateStep secondStep>
    toDtFrac = 1.0
    updaters = [yeeFaraday]
  </UpdateStep>
```

7.3. Text-based (.pre) Input File Structure 93

```
    updateStepOder = [firstStep secondStep]

</MultiField>
```

The `MultiField` we defined ended up having the same updaters as an electromagnetic field created using `EmField`. However, this need not always be the case, as a large number of `FieldUpdater`, `UpdateStep`, and other related objects exist for describing non-conventional systems.

If using the MultiField block in an electrostatic simulation, you must incorporate the linearSolveUpdater block. For more details, visit the linearSolveUpdater sections in VSim Reference.

EM Field in Cylindrical Coordinates

When working in cylindrical coordinates, one must use a special FieldUpdater that is designed specifically for the CoordProd grid. In the list of available updaters (found in FieldUpdater), the updaters which work with the CoordProd grid will be named ending in *CoordProd*.

7.3.9 Particles

In order to define particles in your simulation, you will need to add each type of particle in its own Species block, which will in turn include nested blocks that describe the particles' sources and sinks.

Species Block

The Species block begins by stating the kind of the particle, which takes into account the anticipated dynamic behavior of the particle (relativistic or non-relativistic, for example) as well as the simulation environment it will live in (Cartesian vs. cylindrical coordinate system, electromagnetic vs. electrostatic, etc.). More information on the types of particle kinds can be found in the Species section of VSim Reference.

Also included in the basic Species block are parameters that describe the general characteristics of the particle, such as mass and charge, as well as the parameters that will be used for calculation of macroparticles (`nominalDensity` and `nomPtclsPerCell`).

For a more thorough description of the Species block, please visit VSim Reference: Species.

Macroparticles

- macroPtclWeight used in VSim but not present in .pre file; determines particle resolution in simulation

- **ParticleSource block: VSim determines # of macro particles to try and load in** each cell, each time step. May be << 1

- **applyTimes parameter of said block sets an interval in seconds. VSim tries to** load # of macros ^^ into each cell, each time step, for every time step falling w/in `applyTimes`

- can use `applySteps` instead of `applyTimes`

VSim uses macroparticles to model the kinetics of physical particles. The relationship between the physical density one wishes to model and the density of macroparticles used in the simulation takes the form:

$$numPtclsInMacro = \frac{nominalDensity \times cellVolume}{nomPtclsPerCell}$$

where

- numPtclsInMacro = number of physical particles in a macroparticle

- nominalDensity = nominal density of physical particles

- cellVolume = volume of a simulation grid cell (or average volume over all cells, if nonuniform)

- nomPtclsPerCell = number of macroparticles in a simulation grid cell

The number of physical particles in a macroparticle can be set directly by including the following in the **Species** block:

```
numPtclsInMacro = (float or integer value)
```

It can also be determined by setting both of the following parameters in the **Species** block:

```
nomPtclsPerCell = (float or integer value)
nominalDensity = (float value)
```

The sole purpose of the `nominalDensity` and `nomPtclsPerCell` parameters in the top level of a **Species** block is to calculate the number of particles in a macroparticle of that species, according to the relation given above.

Alternatively, one can define this quantity directly using the `numPtclsInMacro` parameter. With the number of particles in a macroparticle thus determined, one can then load the macroparticles into the simulation domain using one or more `ParticleSource` blocks (a sub-block within the **Species** block); these blocks should be configured to load macroparticles in a manner consistent with the desired physical particle density.

Many types (kinds) of `ParticleSource` can be used within VSim; the complete list is given in the ParticleSource section of VSim Reference.

Use of STFunc blocks to modify density

A number of STFunc input blocks can be used to control or modify the density of physical particles represented by the simulation. Not all `ParticleSource` kinds permit the use of such blocks.

For more information, please visit the section on xvLoaderEmitter in VSim Reference.

Particle Sources

ParticleSource blocks must be nested in the Species block of the particle you'd like to load or emit.

- certain blocks like xvLoaderEmitter will have sub-blocks (PositionGenerator and PositionGenerator)

- certain blocks can add in STFunc blocks (nested) for density functions over time, etc

Particle Sinks

Full Species Block Example

Below is an example of a `Species` block that includes nested `ParticleSource` and `ParticleSink` blocks. For more blocks that are available for nesting under the `Species` block, please visit VSim Reference.

```
<Species electrons>
  kind = nonRelES
  charge = ELECCHARGE
  mass = ELECMASS
  nominalDensity = 1.0e12
```

(continues on next page)

```
  nomPtclsPerCell = 10
  fields = [esField]

  <ParticleSource electronSource>
    kind = xvLoaderEmitter

    # Loading particles uniformly across entire grid
    <PositionGenerator initialPosition>
      kind = gridPosGen
      <Slab initSlab>
        lowerBounds = [0 0 0]
        upperBounds = [NX1 NY1 NZ1]
      </Slab>
    </PositionGenerator>

    # Simple velocity generator, with VEL as velocity variable
    <VelocityGenerator initialVelocity>
      kind = beamVelocityGen
      vbar = [VEL 0. 0.]
      vsig = [0. 0. 0.]
    </VelocityGenerator>
  </ParticleSource>

  # Removing particles at upper y-boundary but saving them for use by other code␣
→components
  <ParticleSink electronSink>
    kind = absAndSav
    minDim = 1
    lowerBounds = [-1 NY -1]
    upperBounds = [NX1 NY1 NZ1]
  </ParticleSink>

</Species>
```

For more details on the xvLoaderEmitter particle source, or any other particle source options, please visit the Particle Sources in VSim Reference.

Particle Tracking

Since VSim is a particle-in-cell code, the position of a specific particle in the particle data structure will vary as the simulation proceeds due to sorting and the creation and/or deletion of particles. Before you can track a particle, you must identify the particle by tagging it.

Note: To track an individual particle, you must use a kind of particle species that enables tagging of individual particles. Creating a species with the kind parameter set to one of the following kinds enables you to create tagged particles that you can then use to generate particle position data for all tags (indicating up to some maximum tag value) by using History.

- relBorisTagged

- relBorisVWTagged

- relBorisVWScale

Before particles can be tracked they must be tagged. There are four ways to tag particles.

- By setting the tags manually with the `manualSrc`.

- By using the tagGen **STFunc** in the `funcVelGen` **VelocityGenerator**. See VelocityGenerator in VSim Reference.

- By using the `fieldScaleVelGen` **VelocityGenerator**. This should only be used with `relBorisVWScale` particles.

- By using the `overwriteTag` in the **Species** block. When this flag is set to true the species takes reponsibility for generating the flags. This is the only option that will work with restore.

If the tags are set manually then the user must insure that the tags are unique or errors will occur tracking the particles.

Example of Using manualSrc to Tag Particles

```
<Species electrons>
    kind = relBorisTagged
    # constants defined previously
    charge = ELECCHARGE
    mass = ELECMASS
    # Place a single particle in the macroparticle
    emField = constMag
    numPtclsInMacro = 1.
    #
    # If the manual source is used to create
    # tagged particles, the tag must be a unique
    # integer or the particle tracking will not
    # work.
    #
    <ParticleSource SingleParticleSrc>
        applyPeriod = 0
        kind = manualSrc
        # manual particle loaders always require
        # AT LEAST 3 velocity components regardless
        # of NDIM; in this case, NDIM=2
        p1 = [R_GYRO 0. 0. V_GYRO V_Z 0.]
        p2 = [0. R_GYRO -V_GYRO 0. V_Z 1.]
        p3 = [-R_GYRO 0. 0. -V_GYRO V_Z 2.]
        p4 = [0. -R_GYRO V_GYRO 0. V_Z 3.]
    </ParticleSource>
</Species>
```

Example of Using tagGen to Tag Particles

```
<VelocityGenerator velGen>
    kind = funcVelGen

    # This sets the tags to be unique integers so they
    # can be tracked by the tagged particle tracking
    # history
    <STFunc component3>
    kind = tagGen
    </STFunc>
</VelocityGenerator>
```

After you have tagged those particles you would like to track using you can record the particles position and/or velocities (or other internal variables such as the weight) by using the rm-speciestracktag **History**.

Example History Block Used to Record Data from Tagged Particles

```
#  Tagged particle trajectory history.
<History trajectory>
  kind = speciesTrackTag
  # Any particle with a tag greater than or  equal to
  # the maximum tag will not be tracked.
  maximumTag = 4
  xComponents = all
  species = [electrons]
</History>
```

For more information on histories, please visit the History section in VSim Reference.

7.3.10 Fluids

One can add charged and neutral fields to your simulation by employing the `Fluid` block. It supports the following three `Kind` options:

- `coldRelFluid`
- `eulerFluid`
- `neutralGas`

A `gasKind` must then be specified, either by using a built-in value or by importing data from an external file. An `InitialCondition` nested block must also be included, as it declares necessary boundary conditions and initial condition parameters.

Below is an example of a fluid being implemented in a simulation:

```
<Fluid neutralFluid>
  kind = neutralGas
  gasKind = Ar
  <InitialCondition >
    kind = constant
    # Lower and upper bounds with respect to the grid
    lowerBounds = [0 0 0]
    upperBounds = [50 50 50]
    # Here, amplitude refers to the gas density, in kg/m^3
    amplitudes = [1.78]
    components = [0]
  </InitialCondition>
</Fluid>
```

More details on `Fluid` and its parameters can be found in the Fluid section of VSim Reference.

7.3.11 Collisions

Reactions

In the Reactions framework, interactions are set up with two required blocks: a `RxnsProcess` block which is used to set the products and reactants for an interaction, and a `RxnPhysics` block which sets the physics (cross-sections/rate

and type of interaction). The third, optional block, the `RxnProcessSettings` block, may also be included if desired.

The example code block at the bottom of this page shows the complete setup of an electron attachment process in which an electron attaches to a neutral argon atom to create a negative argon ion. The optional `RxnProcessSettings` block comes before all other blocks. Then the `RxnsProcess` block sets which species will react, sets the product species that will be created, and points to a `RxnPhysics` block. The cross-sections for determining the likelihood of an attachment reaction are imported via a 2-column data file in the `RxnRate` block within the `RxnPhysics` block. Finally, the `RxnProductGenerator` block determines form of the reaction. The `RxnProductGenerator` should be thought of as what sets the form of the chemical reaction to be undergone by the reactants.

More details of these blocks can be found on the following pages under the Reactions section of the VSim Reference section on Text Setup: RxnProcessSettings Block (optional), RxnProcess Block (required), and RxnPhysics Block (required).

Multiple `RxnsProcess` blocks can point to one `RxnPhysics` block. So, for example if a simulation is to include multiple electron attachment reactions with the same cross-section, then there must be as many `RxnsProcess` blocks as unique reactions, but all may point to the same `RxnPhysics` block.

The order in which the reactants and products are written in a `RxnsProcess` block is important and is determined by which `RxnProductGenerator` is used in the `RxnPhysics` block pointed to by the `RxnsProcess` block. When order is important, the necessary orders are indicated in the the documentation for each ProductGenerator (see the RxnProductGenerator Sub-Block section under the page for the RxnProcess Block: RxnProductGenerator Sub-Block).

In the Reactions framework, reactants and products can be any kinetically modeled particle species or background gas. So the distinction between particle-fluid or particle-particle made in the MCI framework does not exist for Reactions.

```
<RxnProcessSettings RxnProcessSettings>
  updateOrder = random
  updatePeriod = [ 1 ]
</RxnProcessSettings>


<RxnProcess electronAttachmentParticlesRXN0>
  kind = collisionProcess
  reactants = [neutralArgon electrons]
  products = [ArMinus]
  rxnPhysics = electronAttachmentParticlesRXN0electronAttachment
  verbosity = 127
</RxnProcess>

<RxnPhysics electronAttachmentParticlesRXN0electronAttachment>
  kind = generalCollision

  <RxnRate rxnRate>
    kind = twoColumnFile
    crossSectionVariable = velocity
    file = 2ColumnData.dat
  </RxnRate>

  <RxnProductGenerator productGenerator>
    kind = electronAttachment
    thresholdEnergy = 1.0
  </RxnProductGenerator>

</RxnPhysics>
```

Monte Carlo Interactions (DEPRECATED)

There are three main categorizations of interactions recognized within the Monte Carlo Interactions package:

- **Null (non-kinetic)** Interaction not dependent on the full initial state (position and/or velocity) of any particle. These are forced into an occurrence probability of 1, meaning that they occur every time step.

- **Unary (partially-kinetic)** The probability of occurrence and final state are dependent on only one of the particles' full initial states. These interactions are randomly occurring, and an example would be the ionization of a gas by an incident particle.

- **Binary (fully-kinetic)** Both particles' full initial states determine the probability of occurrence and final state of the interaction. These are also randomly occurring, and an example would be the collision of two kinetically modeled particles.

Impact collisions then include the following:

- Elastic scattering collisions

- Exciting collisions

- Ionizing collisions

- Charge exchange

Data for cross-sections used in collision calculations either is built-in to VSim, or can be loaded through various methods. Please see the Monte Carlo Interactions Introduction in VSim Reference for more information on interactions and cross-sectional dependencies. The Using Cross Section Data in that same manual also may be helpful for your simulations.

For more information, please visit the MonteCarloInteractions section of VSim Reference.

Partially- and non-kinetic interactions can be simulated by using a Fluid. For more details on fluids, please visit the Fluid section of VSim Reference.

7.3.12 Histories

You may want to collect specific data from your simulation that is more complex or just different than the default data found in the *Visualization* window. In this case, you can add a `History` block at the end of your simulation.

An example of a `History` block in action is below:

```
<History pseudoPotential1>
  kind = pseudoPotential
  field = multiField.E
  referencePoint = []
  measurePoint = []
</History>
```

This particular history can be used to measure the "voltage" between two points in a simulation, where the points are given in coordinate form.

You can use any number of histories in your simulation, provided that they match the type of simulation you are running (i.e. a history concerned with the magnetic field will not work in an electrostatic simulation, etc.).

For a full list of available histories available in VSim, please see the *Histories* section of the *VSim Reference Manual*.

7.3.13 Macros

The great flexibility of the VSim input file languages allows VSim to be used to model a wide variety of systems, but at the same time can make the construction of input files rather daunting. Macros simplify input file construction, and are a mechanism to abstract complex input file sequences into (parameterized) tokens. In its simplest form, a macro provides a way to substitute a code snippet from an input file. A user can then put only the macro into the input file, and it will be expanded into the full input file at the time the .pre file is preprocessed.

An example of a macro has already been used in the *Geometries* section of this document, wherein we created a hollow cylinder:

```
<macro hollowCylinder>
  H(-INNER_RADIUS^2 + x^2 + y^2)*H(RADIUS^2 + x^2 + y^2)
</macro>
```

Macros can contain your own functions, built-in functions from VSim (like geoBoxP, for example), or a combination of the two types. Once your macro is defined, you can call it as many times as you wish in your simulation.

VSim also contains a number of pre-defined macros that are used throughout the example input files available through the VSimComposer interface. You may find the VSim macros to be helpful in your own simulations, especially in cases like `addFarFieldBox` where the macro automatically adds 18 different (and necessary) histories to your simulation.

The ability to use certain macros in VSim is tied to the particular VSim license in use.

For further information on macros, please visit either VSim Customization for more of the basics of macros, or refer to VSim Reference for a full list of macros included in VSim versions 7 and 8.

7.3.14 XSim Block

When you open a .pre file, the first thing you'll see is a section containing general information, including the .pre file name, the example version, and copyright information.

The next section will be the XSim block. See Fig. 7.6.

The sections in the XSim block are as follows, and appear in Fig. 7.7.

1. **image** - The image parameter gives the name of the picture, located in the same directory as the input file, that will be displayed on the right hand side of the *Editor* pane in the **Setup** tab. Frequently, this image is used to illustrate key parameters such as dimensions of a physical structure. 400 x 500 pixels is a good image size.

2. **longDescription** - This text block will be visible above the image. It's generally used to give a description of what the simulation does and what will happen when key parameters are modified.

The next four parameters are only really useful to very advanced users who are creating and placing input files in the Examples directory of VSimComposer. The Examples directory can be found in [VorpalInstallDirectory]/Contents/Examples. Items 3, 4, and 5 are depicted in Fig. 7.8.

3. **thumbnail** - This is the small image that is visible when you select an example, and is located in the same directory as the input file. 250 x 250 pixels is a good image size.

4. **shortDescription** - This is the name given to the example file.

5. **description** - This is the description given in the *Examples* window.

6. **analyzers** - VSimComposer will load the analysis script specified in brackets on the left hand side (which should be located in the same directory as the input file) for use in the **Analyze** window.
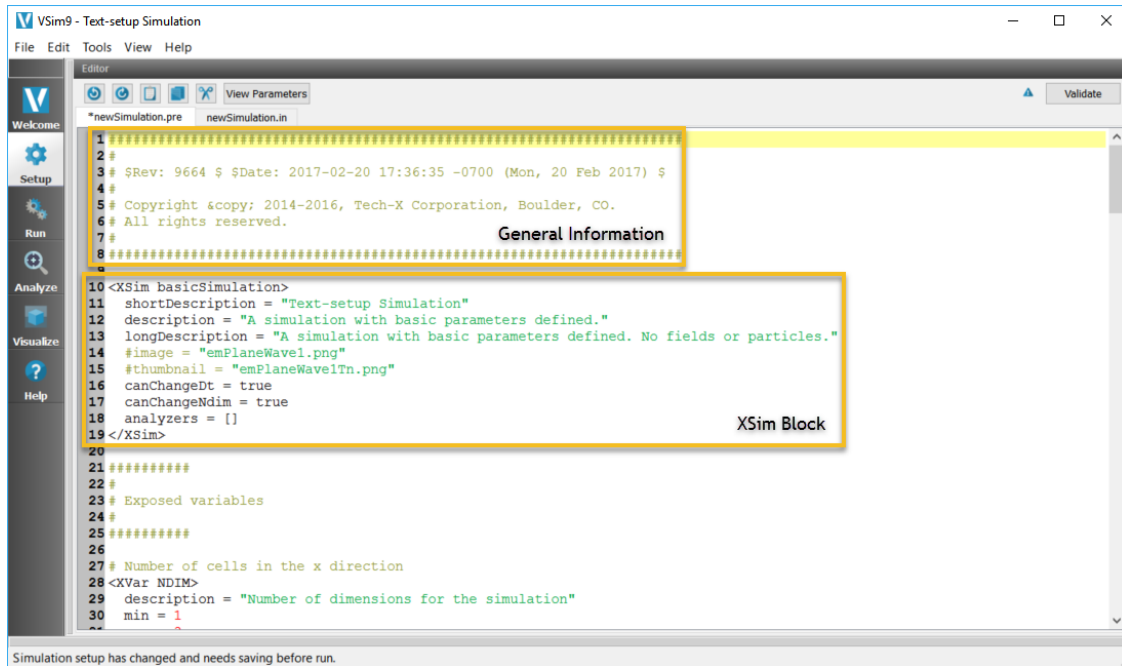
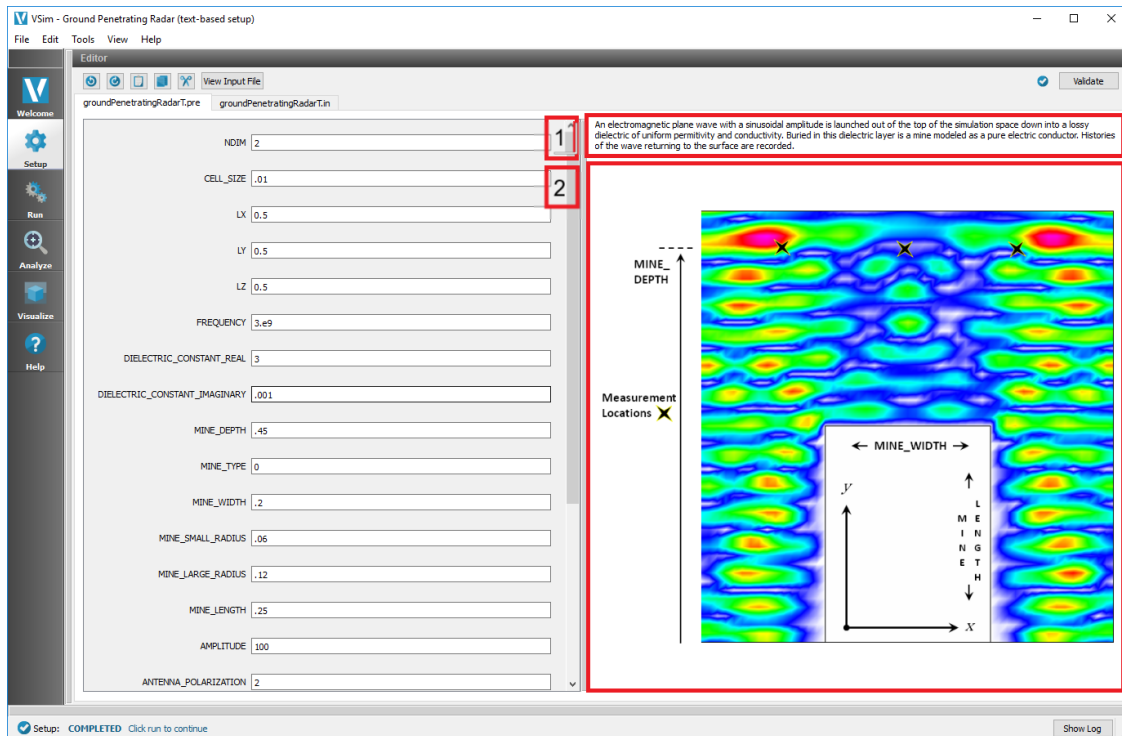Fig. 7.6: General information and XSim block

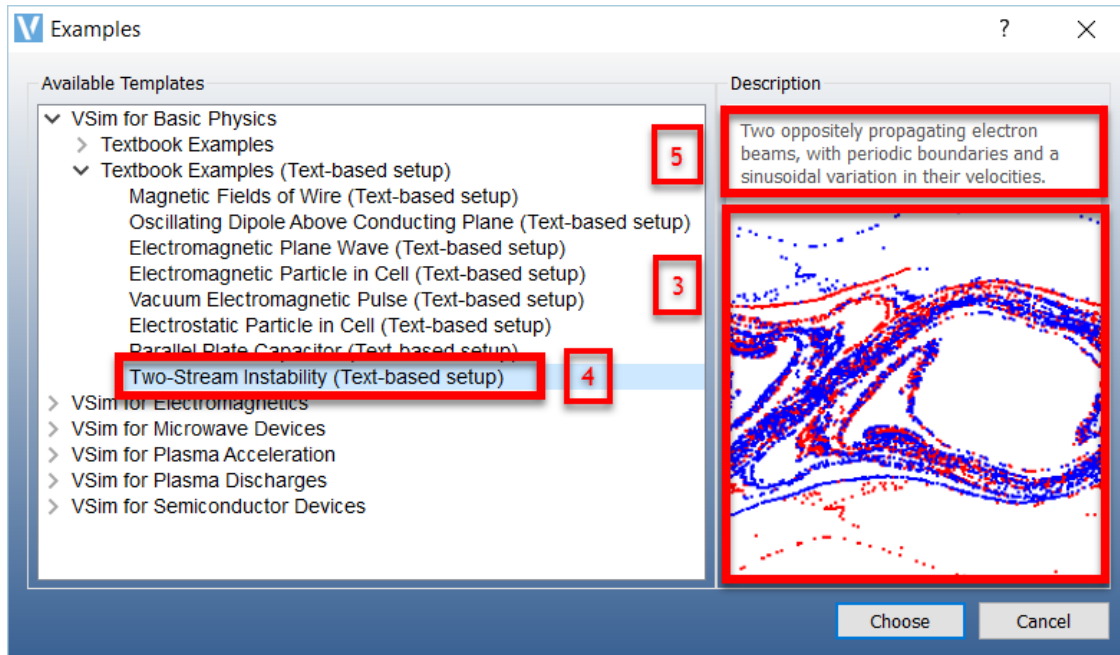Fig. 7.7: The image and longDescription in the Parameters View.

Fig. 7.8: Select an Example window

7.3.15 XVars

Exposed Variables

You can set which parameters are "exposed" in the Key Parameters view through adding or removing XVar blocks. The following is an example of an XVar block:

```
<XVar variableName>
  description = "Description of the variable"
  min = minimum value
  max = maximum value
</XVar>
```

The lines in this block are as follows:

- **<XVar variableName>** Begins the XVar block for variableName. The variable name here must exactly match the variable that you are trying to define.

- **description** Describes the variable and will appear when the cursor is placed over the variable name in the Key Parameters view.

- **min** This is the minimum value for the variable and is optional. Setting a minimum can be very useful with certain simulation parameters such as cell size, which can cause an instability if incorrectly specified.

- **max** This is the maximum value for the variable and is optional.

Note: The name of the key parameter will turn red if there is no value given for the parameter, or if the parameter is not greater than or equal to **min** and less than or equal to **max**, if they are specified.

Primary Variables

The Primary Variables are variables that correspond to the XVar blocks defined in the above Exposed Variables section. The syntax for setting primary variables is the same as setting any other user-defined variable:

```
$ VARIABLENAME = default value
```

Variable names are generally in all capital letters. If you redefine this value through the Key Parameters pane, it will overwrite the default value set in the .pre file.

Setting Key Parameters

You can also declare your primary variables right before their respective XVar blocks and essentially condense the Exposed Variables and Primary Variables sections into one, as shown below:

```
$ VARIABLEONE = default value
<XVar VARIABLEONE>
  description = "First variable"
  min = minimum value
  max = maximum value
</XVar>

$ VARIABLETWO = default value
<XVAR VARIABLETWO>
  description = "Second variable"
</XVar>
```

The parameters defined in the sections above then can be passed as the values for global variables that define your simulation. For example, if you defined a variable named NDIM as below:

```
$ NDIM  = 3
<XVar NDIM>
  description = "Number of simulated dimensions"
  min = 3
</XVar>
```

You could then use this parameter as the value for the global variable dimension.

```
dimension = NDIM
```

For more information on global variables, see either globalvariables or the section on global variables in VSim Reference.

EXECUTING THE COMPUTATIONAL ENGINE (VORPAL)

8.1 Running Vorpal within VSimComposer

8.1.1 Run Window

Select the Run Icon

Once your validation is successful, in the upper right corner you will see a checkmark by the *Save and Setup* button. You can now select the **Run** icon from the icon panel on the far left of the VSimComposer window.

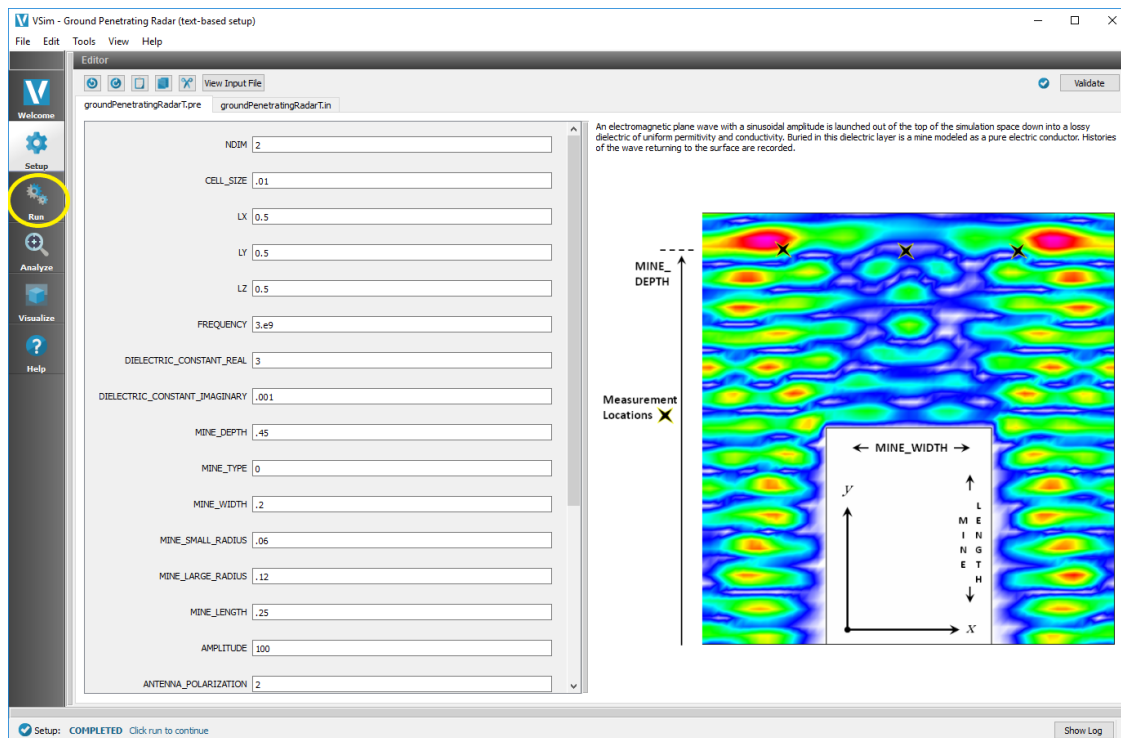Click on the **Run** icon as shown in Fig. 8.1.

Fig. 8.1: Run Icon in Icon Panel

The Run Window

The VSimComposer Run window contains two panes. As displayed in Fig. 8.2, the *Runtime Options* pane is on the left and the *Logs and Output Files* pane is on the right, which contains an *Engine Log* and a *File Browser* tab.
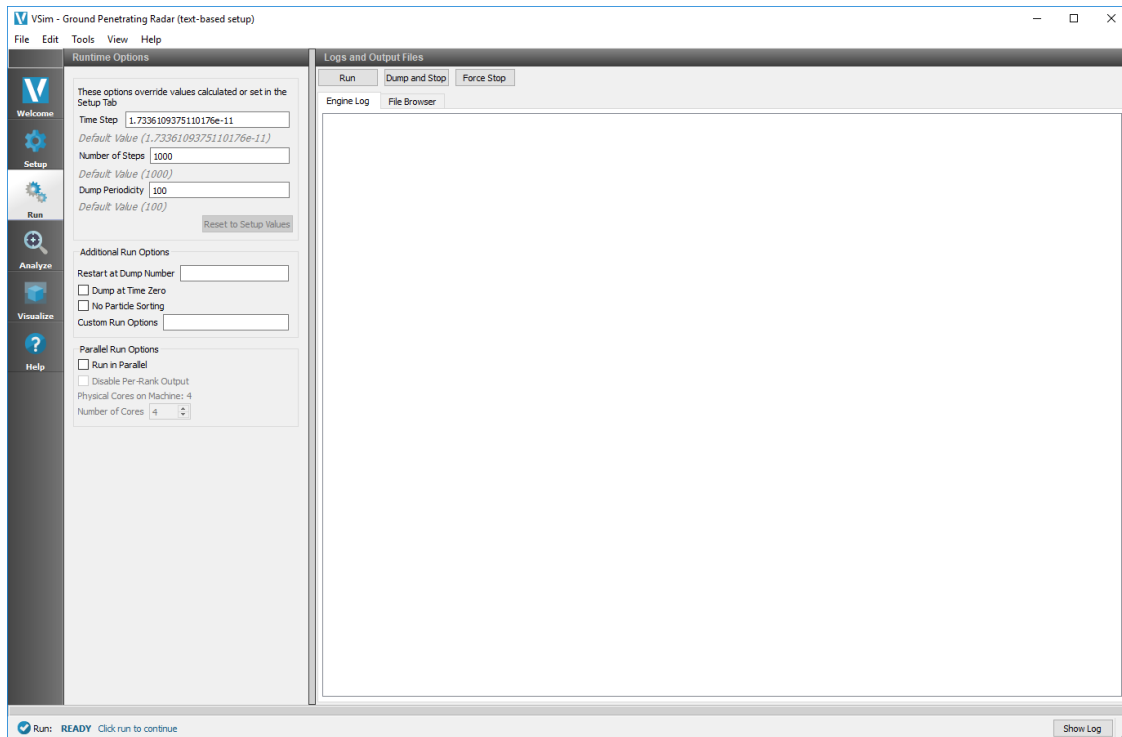

Fig. 8.2: The Run Window

Runtime Options

VSimComposer enables you to specify runtime options, and in some cases, override the settings in your simulation input file. The *Runtime Options* pane contains fields and options that give you the flexibility of command line control with the convenience of a graphical interface much like the key input parameters of the setup window.

Time Step (s): Step size to use in the simulation. When choosing the step size for your simulation, you must consider stability requirements. For example, for electromagnetic simulations, you should specify a step size that satisfies the Courant condition [CFL28]. You can override the **dt** variable using the −dt command line option. You must define *dt* in an input file.

Number of Time Steps: Number of steps to take. (In the case of a restart, **nsteps** is the number of additional steps.) This can be overridden with the −n command line parameter. You must define *nsteps* in an input file or on the command line.

Dump Periodicity (time steps): How often to dump the data; indicates data is to be dumped whenever the time step has increased by this amount. The command line parameter -d overrides this variable. Must have this defined in an input file or on the command line.

If you make changes to the *Runtime Options*, you can restore the options to their original settings by clicking on the *Reset to Setup Values* button located below the *Dump Periodicity* in the left pane.

Additional Run Options

Restart at Dump Number: It is possible in this menu to restart a previous simulation with the *Restart at Dump Number* field. VSim will load all the data associated with the dump number input in the field, and continue running the simulation from that time.

Dump at Time Zero: Dump data at start of simulation, before any time has passed. This option is useful for debugging purposes. It lets you see whether Vorpal used the data you wanted it to use at the start of the simulation.

No Particle Sorting: This passes the -ns flag to Vorpal, which tells Vorpal to not sort the particles. Sorting the particles (does not work with cell species) can affect the performance of your simulation.

Custom Run Options: You can pass command line arguments to the Vorpal engine here. The list of options can be found in the Reference Manual under Vorpal Command Line Options.

Parallel Run Options

Run in Parallel: You may run your simulation in parallel as multiple processes by checking this box. Then, select the number of cores you want to run on. You cannot run on more cores than you are licensed for. The contents of the Engine Log are dumped into separate comms files when you run in parallel. You can check the *Disable Per-Rank Output* box if you don't need these files. Additional information on running in parallel can be found below in *Running in Parallel from VSimComposer*.

Run the Simulation

For our example, we'll run this simulation using only the default existing settings from the input file.

You do not need to select any file in particular in the *File Browser* tab before clicking on the *Run* button. However, if the *File Browser* tab display area is too narrow for you to see the full file names in the filename list and you would like to see the file name extensions of the files in the file browser, you can adjust the width of the filename field by using your mouse to drag the column border.

Click on the *Run* button at the top of the *Logs and Output Files* pane as shown in Fig. 8.3.

Stopping the Simulation

VSimComposer features the ability to *Dump and Stop* and *Force Stop* a simulation. The buttons for these actions are located next to the *Run* button. The two actions have slightly different uses. The *Dump and Stop* button is to halt a simulation that is running normally to free up the processors used for another task, or so that one may vary the parameters and restart. When a simulation is stopped it will dump all field and history data, so that it may be restarted from the same point later. The output of a successfully stopped simulation is shown in Fig. 8.4.

The *Force Stop* is to be used if you realize that an error was made in the input file after clicking *Run* and needs to be corrected. If *Force Stop* is used the field and history data will *NOT* be written to a .h5 file before the simulation stops, but it will stop the simulation immediately rather than exiting gracefully. This option is particularly useful when you may have more mesh cells than you intended, where the simulation is trying to allocate more memory than you intended. The output of a successfully force stopped simulation is given below.

Restarting a Simulation

With VSimComposer it is possible to restart a simulation that has previously been run. This is useful if it is desired to add more time steps to the initial simulation, or if the simulation had been stopped in the middle of the run. Underneath the *Runtime Options* pane of the run window, under *Additional Run Options*, there is a *Restart at Dump Number* field.

Simply put in the last memory dump of the simulation and then click on the *Run* button, like running a normal simulation. This process is demonstrated in Fig. 8.6.
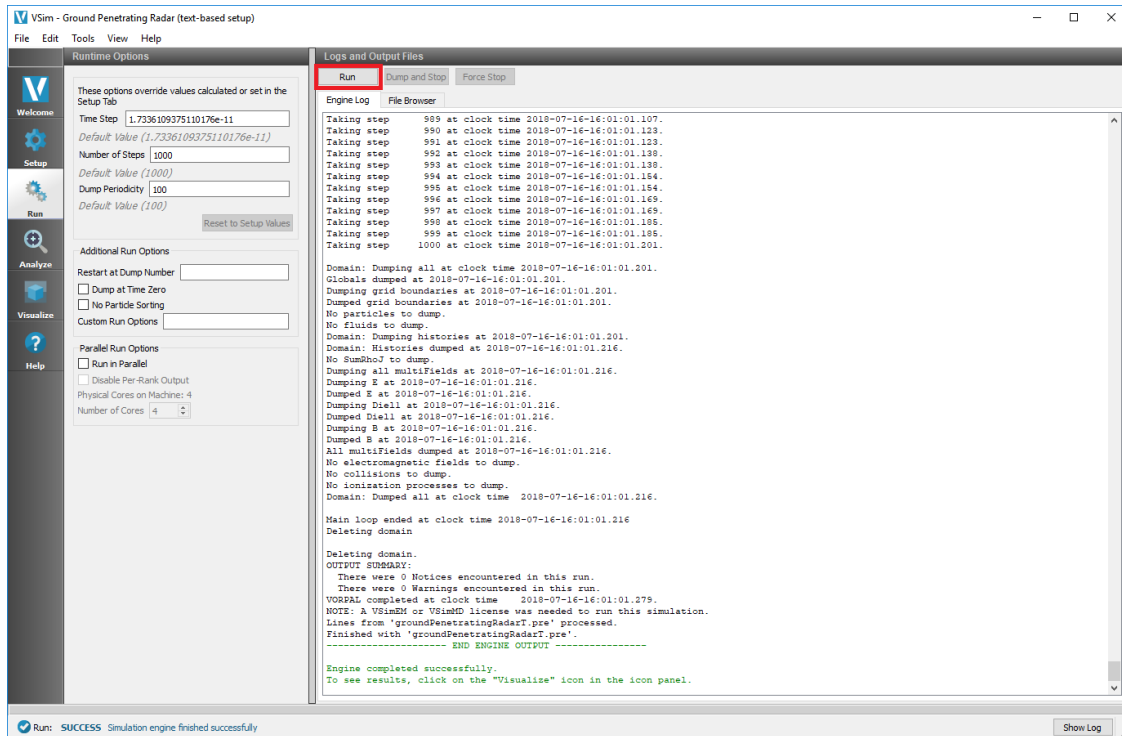
Fig. 8.3: Run Button

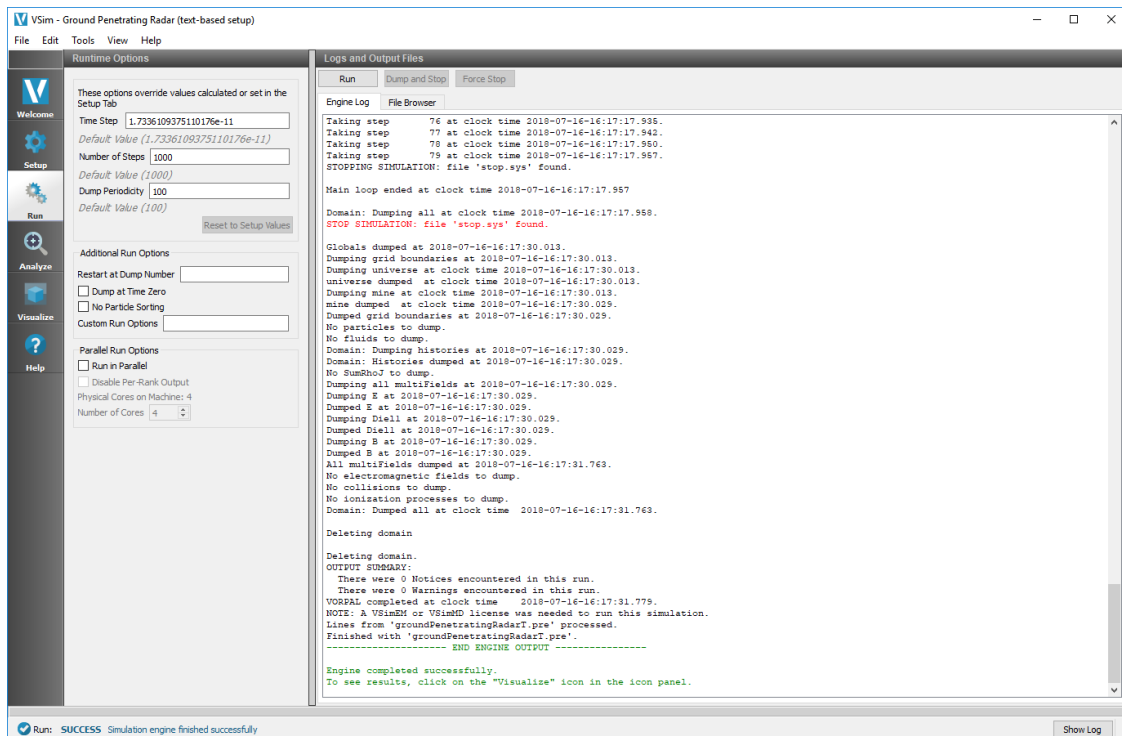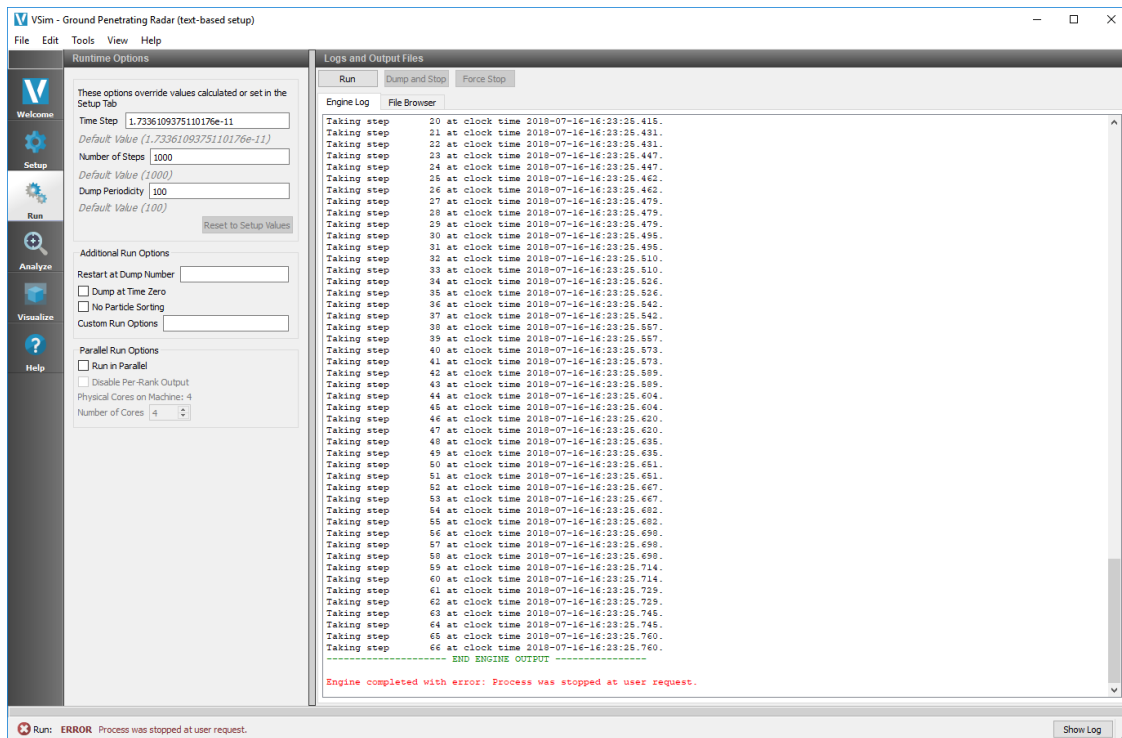Fig. 8.4: Stopped Simulation

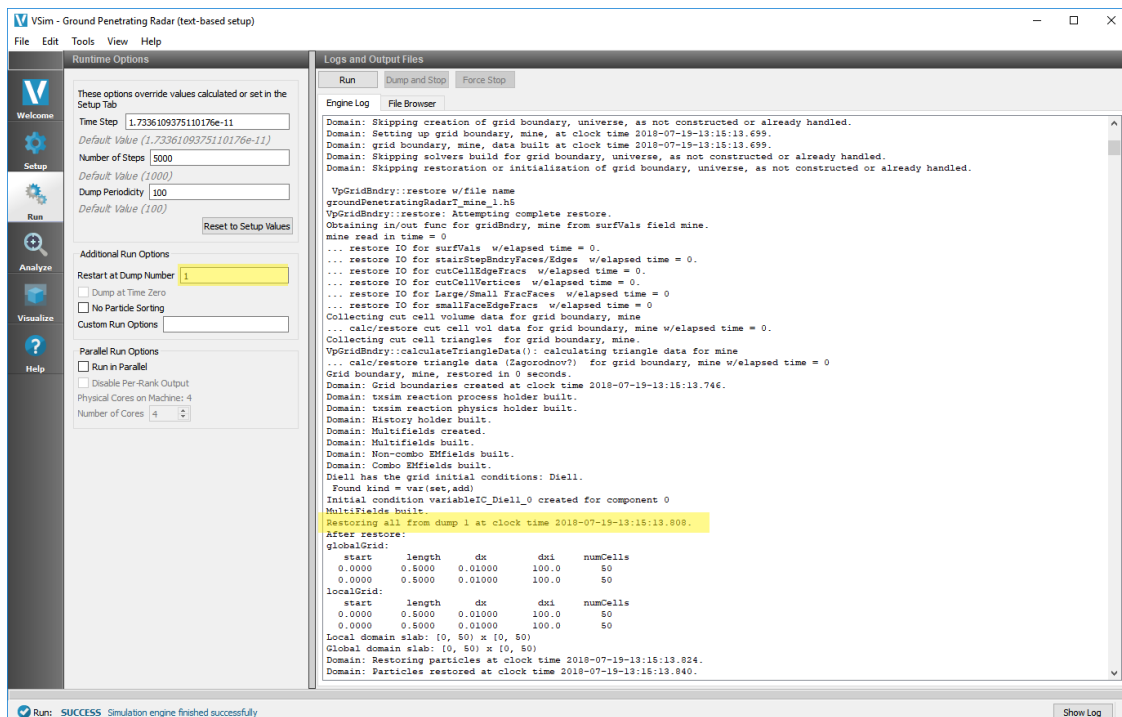Fig. 8.5: Force Stopped Simulation

Fig. 8.6: Restarting a Simulation

View the Engine Log

VSimComposer notifies you of the progress of its activity by reporting results along the way in the *Engine Log* tab as shown in Fig. 8.7.
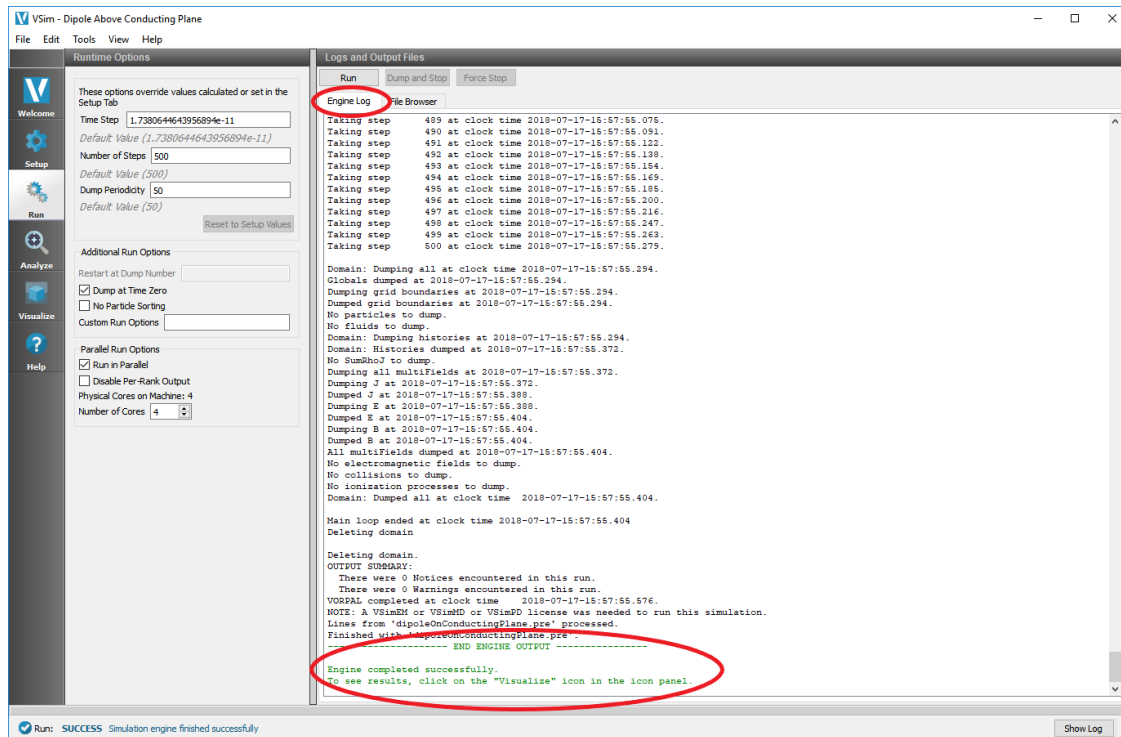
Fig. 8.7: Engine Log

File Browser Tab in the Logs and Output Files Pane

In previous steps, the *File Browser* tab was located behind the *Engine Log* tab in the *Logs and Output Files* pane. Click on the *File Browser* tab to bring it to the front as shown in *File Browser Tab in Logs and Output Files Pane*.

As with the *File Browser* in the *Setup* window, the *File Browser* in the **Run** window also has the *Smart Grouping* and *All Files* pull-down menus at the bottom of the tab.

After a simulation has been run, you will be able to see the files that were output based on your number of time steps and the dump periodicity. See Fig. 8.8.

Output File Naming Conventions

The first part of the output file name is the name of the input file. This is often referred to as the *base name*.

The second part of the output file name indicates the file's contents, for example:

- The name of the field or particle species, such as E, B, or electrons.

- Globals for the file containing global variables such as the global grid, needed for restarts.

- History, containing data recorded over time.

- comms and "all" text files, containing various debugging and information about the simulation.
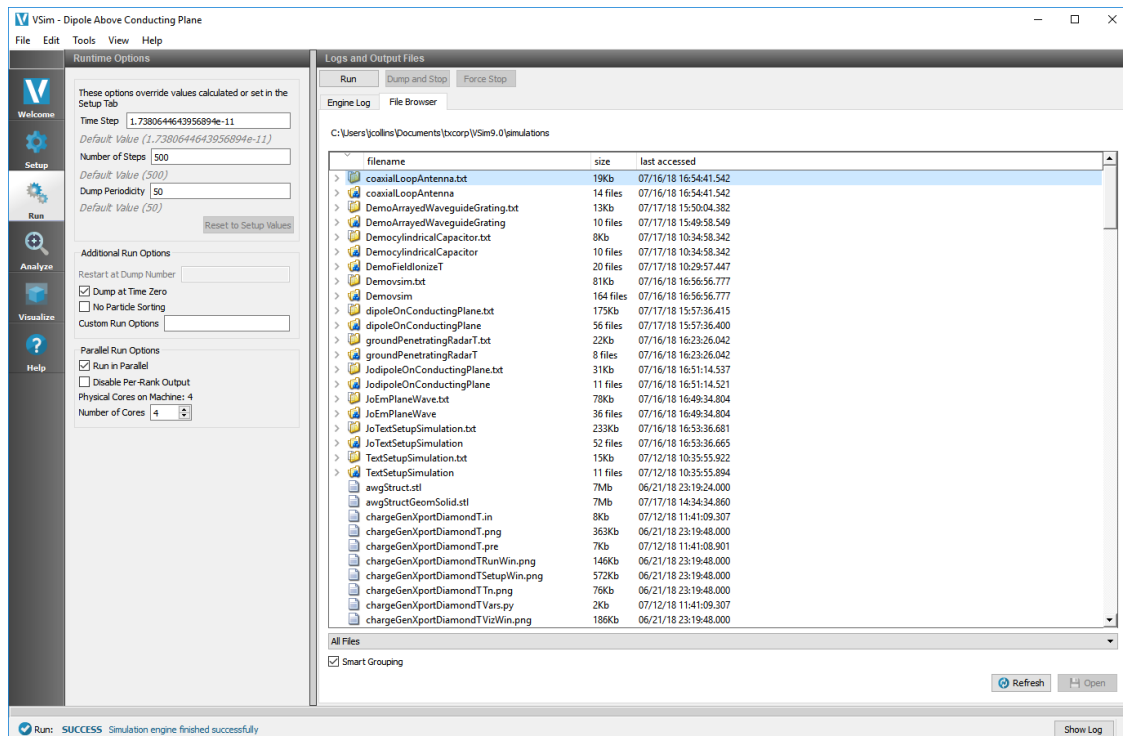
Fig. 8.8: File Browser Tab in Logs and Output Files Pane

The third part of the output file name is the dump number.

The final part of the output file name is the suffix.

8.1.2 Running in Parallel from VSimComposer

VSimComposer runs simulations in serial by default when you open a new simulation. If you are running on a local system with multiple cores, you can run your simulation in parallel as multiple processes.

Permanently Switching the Engine to Parallel Execution

One can switch the engine to parallel execution from within *Tools -> Settings*, by following the steps provided in *MPI*. See *Fig. 8.9*.

Defining the Number of Processors From the Run Window

It is also possible to switch the number of processors the simulation is run on within a single simulation session in the *Run* window. In the *Runtime Options* pane, you will find the *Parallel Run Options* box. Here you can define the number of cores to run on. You cannot use more cores than you are licensed for. The options will appear with their defaulted values for that simulation, but you can override the defaults. This setting will be retained for as long as the current file is open. See *Changing number of cores on a per-simulation basis*.

Now change any command line options as desired or run as usual by pressing the *Run* button.

Fig. 8.9: Switching on parallel execution in the settings tab.

Fig. 8.10: Changing number of cores on a per-simulation basis

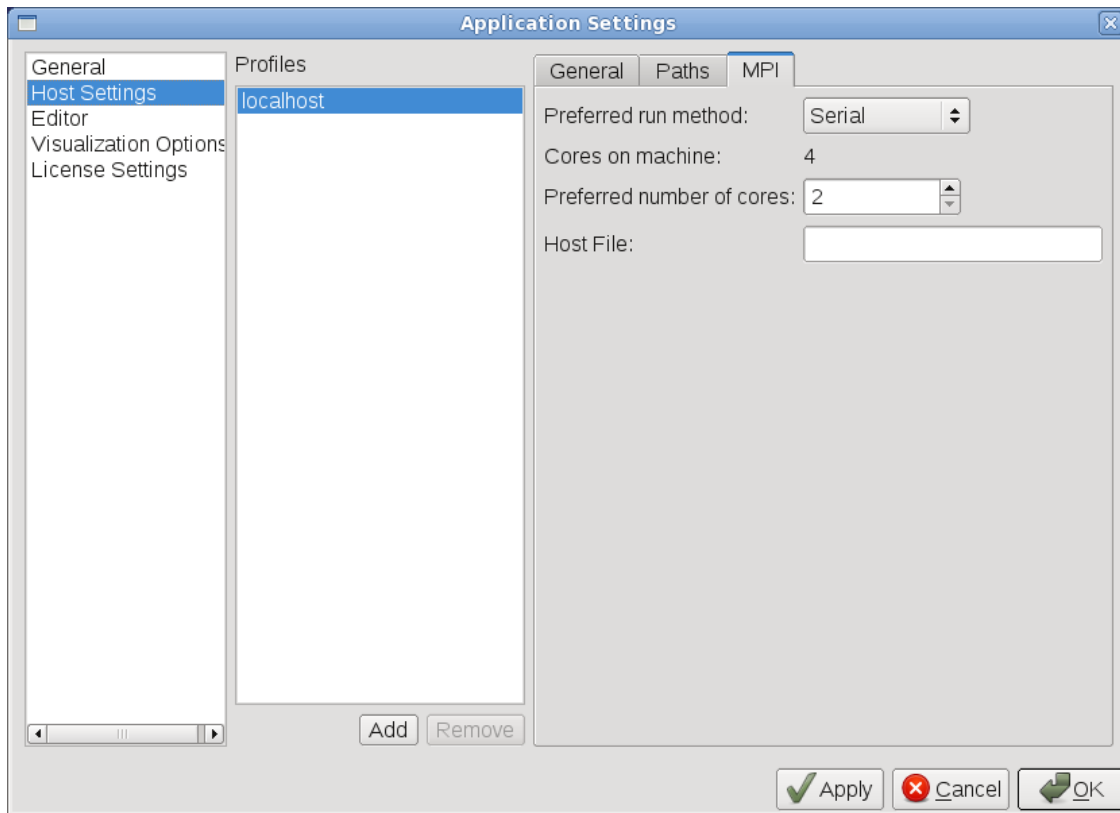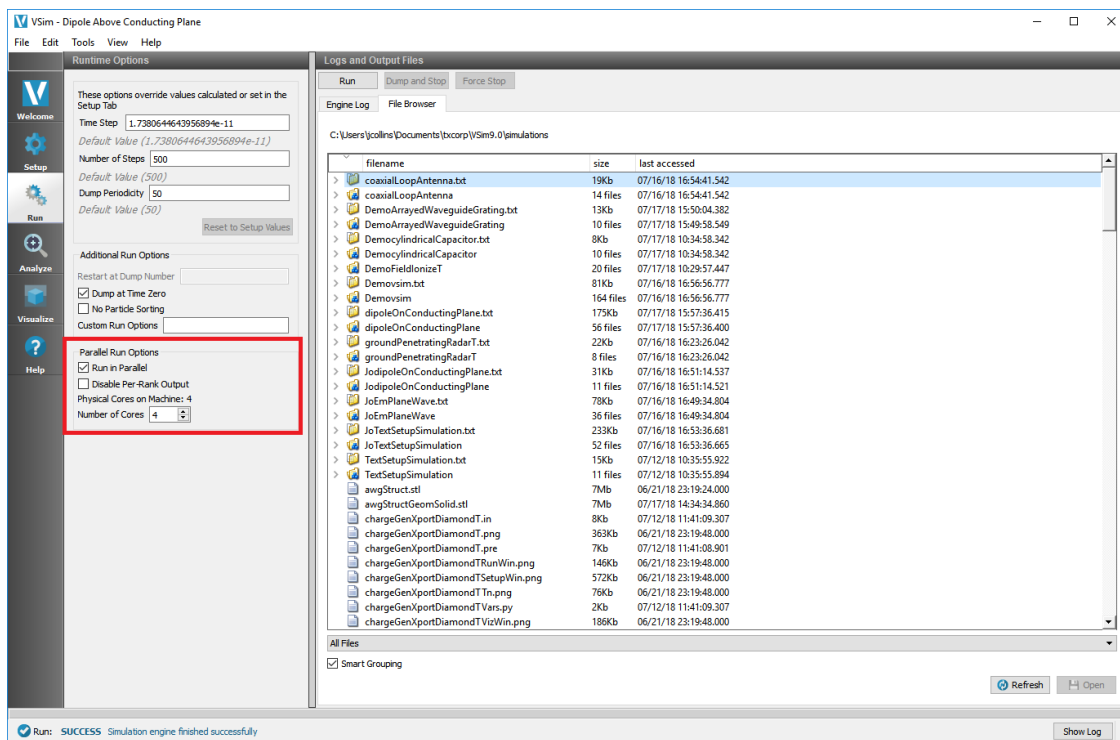## 8.2 Running Vorpal from the Command Line

8.2.1 Running Vorpal From the Command Line

The following sections describe how to run Vorpal from the command line.

Setting Up Vorpal Command Line Environment

Vorpal needs several environment variables set before it can be run from the command line. VSim provides scripts to setup the environment on each operating system.

The following instructions use the variable SCRIPT_DIR which is the directory where VSim is installed. For example, this would be something like `SCRIPT_DIR=C:Program FilesTech-XVSim-10.0` (Windows), `SCRIPT_DIR=/usr/bin/VSim-10.0` (Linux), `SCRIPT_DIR=/Applications/VSim-10.0/VSimComposer.app/Contents/Resources` (Mac).

On Windows

Open a Command Prompt (run cmd.exe) and execute the following line:

```
C:\> %SCRIPT_DIR%\setupCmdEnv.bat
```

On Linux or Mac

In a bash shell, source the VSimComposer.sh script as follows:

```
$ source $SCRIPT_DIR/VSimComposer.sh
```

This is a bash shell script, which means you must be running the bash shell to execute the above command. If you are normally a csh/tsh user, you will need to start up a bash shell to execute the above command and subsequently execute VSimComposer.

Most distribution's operating systems will allow you to add the above command to the .bashrc file in your home directory, which will prevent having to run it each time you log in. Any changes you make to your .bashrc do not take effect until the next time you log in, so after modifying your .bashrc file, you must execute the following command in your current shell, but will not need to do it in the future:

```
$ source ~/.bashrc
```

Serial Computation

The Vorpal executable for use in serial computation is named **vorpalser**. Except as noted, the explanations and tutorials within the *VSim User Guide* and VSim Examples demonstrate Vorpal usage for serial computations. Here is an example of Vorpal command line invocation using an input file named `myfile.pre` and specifying 1000 time steps, outputting the result data (dumping) every 500 steps. By default, the output files for this example would be named using the format `myfile.out`.

```
vorpalser -i myfile.pre -n 1000 -d 500
```

The Vorpal computation engine for serial computations also creates a single text file named `myfile_comms_0.txt` unless this has been suppressed by command line or input file options.

Parallel Computation

The Vorpal executable for use in parallel computation is named **vorpal**. This section explains use of the Vorpal executable program for parallel computations.

Vorpal for parallel computations requires the Message Passing Interface (MPI). On Mac and Windows, you must use our bundled MPI. On Windows the parallel message passing interface (MPI) library provided with **VSim** is MS MPI (from Microsoft). On Mac, the parallel message passing interface (MPI) library provided with **VSim** is OpenMPI. On Linux, the parallel message passing interface (MPI) library provided with **VSim** is MPICH. If there is a reason why you must use a system **MPI**, please contact Tech-X support, who will quote you for a custom installation.

For administrator information about **MPI** for use with Vorpal, see *Running Vorpal from a Queueing System*.

Running Vorpal with mpiexec

In order to run Vorpal in parallel via the command line, you must first add the <VORPAL_BIN_DIR> to your PATH, as noted in the running-vorpal-from-the-command-line-command-line-options section.

To run Vorpal in parallel, execute the following command:

```
mpiexec -n <#> vorpal -i filename.pre
```

in which <#> is the number of processors, **vorpal** is the executable program for parallel computations, and `filename.pre` is the name of the input file (which must be in the current directory, or must be specified by a full path).

Following **mpiexec**, but before **vorpal**, you can specify a variety of **mpiexec** options. In particular, for the openmpi implementation of MPI (supplied with macOS), one may need to add the arguments, *-x PYTHONPATH -x LD_LIBRARY_PATH* to ensure that all processes are using the correct values for these environment variables. For the MPICH implementation of MPI (supplied with Linux) these arguments are not needed, as MPICH by default exports all environment variables to all processes. For more information about **mpiexec**, including the complete list of options, it can be run with **mpiexec -h**.

Following **vorpal**, you can specify a variety of Vorpal options. Some of the more common options are

```
vorpalser -i esPtclInCell.pre -o newesPtclInCell
```

```
vorpalser -i esPtclInCellSteps.pre -r 50
```

For a complete list of options, see VSim Reference: Running Vorpal from the Command Line Options.

If a parameter is both set within the input file and specified on the command line, the command line parameter value takes precedence. The command line override enables you to configure an input file with default values while exploring alternative parameter settings from the command line. From the command line, you can quickly change simulation run lengths, dimensionality, output timing, etc.

Vorpal automatically adjusts its decomposition to match the number of processors it is given, unless a manual decomposition is provided for the correct number of cores in the input .pre file.

In contrast to Vorpal for serial computation, which creates a single text output file, Vorpal for parallel computation creates multiple text output files. Each individual processor from the parallel run sends comments to a different output file. A parallel computation output file's name includes a label that identifies the number of the processor that generated that file, for example:

- `esPtclInCell_comms_0_1.txt`
- `esPtclInCell_comms_0_2.txt`

in which the final _0 and _1 before the file name suffix indicate the number of the processor.

By default Vorpal writes one HDF5 file for each field or particle species even for a parallel run. However, one can modify this behavior, as noted at the above link. Having one file for each field or particle species for each processor can sometimes get around parallel I/O problems. When that is necessary, one can construct a single file for a field or particle species using the utilities, mergeH5Flds and mergeH5Ptcls, which come with Vorpal.

Running Vorpal with mpiexec Using a Hostfile

If the you need to run an MPI job but do not have access to a queuing system then a hostfile must be set up. If this is the case you must know the node names on the cluster that the job is to be run on. You must then create a text file with your text editor of choice, this is your hostfile, and place it in your home directory. The hostfile simply contains each node name repeated on a new line as many times as there are threads in that node. For example consider a two node cluster with four threads each, the hostfile will contain

```
node1
node1
node1
node1
node2
node2
node2
node2
```

To run a job one must then source VSimComposer shell script using the command:

```
source <VSIM_SCRIPT_DIR>/VSimComposer.sh
```

Note: This action changes your environment in your current shell, and so may make other programs fail. Do this in a separate shell from any shell in which you intend to run standard programs, like vi or emacs.

You are now ready to run in MPI using the *mpiexec* command with the above hostfile (signified as <hostfile> below. For MPICH (which is provided for Linux), the command is

```
mpiexec -f <hostfile> -n <#> <other mpiexec options> vorpal \
  -i simulationname.pre <other vorpal options>
```

The equivalent command for openmpi (which is provided for macOS) is

```
mpiexec --hostfile <hostfile> -n <#> <other mpiexec options> vorpal \
  -i simulationname.pre <other vorpal options>
```

The number of nodes, *<#>*, must be consistent with the computational resources and the hostfile.

8.3 Running Vorpal on a Cluster using a Queuing System

8.3.1 Running Vorpal From a Queueing Systems

Running Vorpal with **MPI** either directly or with Parallel Queuing Systems requires use of different shell scripts to enable invocation of the Vorpal executable, as outlined below. In this section we discuss Linux queuing systems. For running Vorpal through Windows HPC Cluster Pack see *Running Vorpal on a Windows HPC Cluster*.

Queuing systems, such as PBS, LoadLeveler, LSF, SGE and Slurm, require the submission of a shell script with embedded comments that act as commands that the queuing system interprets. Below we show some of the more common embedded comments. Discussion of all the embedded comments is beyond the scope of this document. Furthermore, the command for submitting the job can vary. Below we will provide a common command, but you should contact the system administrator to ensure that you have the correction job submission command.

Lustre File Systems

Many supercomputers use the Lustre file system, which has multiple Object Storage Targets (OSTs). Using more of these improves output speed for the large files produced by Vorpal in large-scale computing. The number to be used is set by a stripe command, such as

: lfs setstripe –count C .

which sets the number of OSTs to C in the current directory. As noted at https://www.nersc.gov/users/ storage-and-file-systems/i-o-resources-for-scientific-applications/optimizing-io-performance-for-lustre/, one should set C according to

File size	C
1-10GB	8
10-100GB	24
100GB+	72

PBS/Torque/OpenPBS

Here is an example of a basic shell script for a PBS-based system.

```
#PBS -N vaclaunch
#PBS -l nodes=2:ppn=2
export VSIM_DIR=$HOME/VSim-10.0
source $VSIM_DIR/VSimComposer.sh
cd /directory/containing/your/input/file
mpiexec -np 4 vorpal -i vaclaunch.pre -n 250 -d 50
```

The *-l* commands relate to the resource requirements of the job. This file explicitly specified the number of cores per node, and number of nodes, so we have a total of four MPI ranks on which to execute the job, which is mirrored in the mpiexec *-np* argument.

If the contents of the above file are in *vaclaunch.pbs*, then the job would commonly be submitted by

```
qsub vaclaunch.pbs
```

although some MOAB based systems might use msub.

Sun Grid Engine (SGE)

Here is another example, this time for a SGE (Sun Grid Engine) job.

```
#$ -cwd -V
#$ -l h_rt=0:10:00
#$ -l np=16
#$ -N magnetron2D
export VSIM_DIR=$HOME/VSim-10.0
```

(continues on next page)

```
source $VSIM_DIR/VSimComposer.sh
mpiexec -np 16 vorpal -i magnetron2D.pre
```

This time the *-cwd -V* tells the queue system to use the current working directory for the job, and to import the current environment and make this avialable for the script.

In this case, the queue system calculates the configuration based on the choice of 16 cores. On this cluster the *-l* commands is also used to specify the run duration, which, in this example, is set to 10 minutes.

If the contents of the above file are in *magnetron2D.qsub*, then the job would commonly be submitted by

```
qsub magnetron2D.pbs
```

Platform LSF

Here is a third example, for the Platform LSF system:

```
#BSUB -o EBDP-VSim10.0.out
#BSUB -e EBDP-VSim10.0.err
#BSUB -R "span[ptile=16]"
#BSUB -n 32
#BSUB -J testVSim10.0
#BSUB -W 45
cd $HOME/electronBeamDrivenPlasma
export VSIM_DIR=$HOME/VSim-10.0
source $VSIM_DIR/VSimComposer.sh

export MYJOB=electronBeamDrivenPlasma.pre

mpiexec -np 32 vorpal -i ${MYJOB}
```

With this submission system and job scheduler, *-W* is used to denote the wall time for the job in minutes. The name is passed by *-J*.

Sometimes one must reference a specific project for accounting purposes in the job submission file. For this you may use the *-A* option.

If the above commands are in the file, *electronBeamDrivenPlasma.lsf*, then the job is commonly submitted by

```
bsub < electronBeamDrivenPlasma.lsf
```

Slurm

Here is an example from a Cray system with a custom Vorpal build running from a directory /project/nnnnn/gnu-5.2.40.

```
#!/bin/bash
#SBATCH --account=nnnnn
#SBATCH --job-name=lpa
#SBATCH --output=lpa.out
#SBATCH --error=lpa.err
#SBATCH --nodes=2
#SBATCH --time=00:05:00
srun --ntasks=32 --hint=nomultithread --ntasks-per-node=16 /project/nnnnn/gnu-5.2.40/
↪vorpal-exported/bin/vorpal -i laserPlasmaAccel.pre -n 100 -d 20
```

If the file containing the above is named, *laserPlasmaAccel.slm*, then this job is submitted with

```
sbatch laserPlasmaAccel.slm
```

You can check on your job with

```
squeue -u $USER
```

and you can stop the job with

```
scancel JOBID
```

where JOBID is the job id returned by squeue.

8.3.2 Running Vorpal on a Windows HPC Cluster

Note: Prior to running VSim with Windows HPC Cluster tools, please ensure that VSim is properly installed on your Windows Cluster. (See VSim Installation Guide)

Setting up the Simulation Directory

The following is an example of how to run an example VSim simulation on a Windows Cluster. In this example, the UNC Share Path that is set up on all nodes in the Cluster including the headnode is:

```
\\hpcheadnode\scratch
```

This path should be replaced by the path to your shared drive, whatever it might be. Paths in UNC (Universal Naming Convention) should be used, so \\machine_name\name_of_share\directory instead of S:\directory.

To start, we create a new simulation in a directory on the shared drive. To do this, run VSimComposer on the headnode, which should be installed on the shared drive as Fig. 8.11 shows.

In VSimComposer, follow these actions:

- Select the *New → From Example. . .* menu item in the *File* menu.

- In the resulting *Examples* window expand the *VSim for Basic Physics* option.

- Expand the *Basic Examples* option.

- Select *Parallel Plate Capacitor* and press the *Choose* button.

- Select the text in the *Directory* field and replace it with the shared drive UNC path. See Fig. 8.12 below.

Continue with:

- Select the *New Folder* button and type in *parPlateCapacitor* for the name. See Fig. 8.13 below.

Now:

- Click the *Create* button in the *New Folder* dialog.

- Click the *Save* button in the *Choose Simulation Name* dialog.

- At this point, you would normally change the setup to suit your simulation needs and possibly run it locally to check if you are on the right track, but for now, just exit VSimComposer.
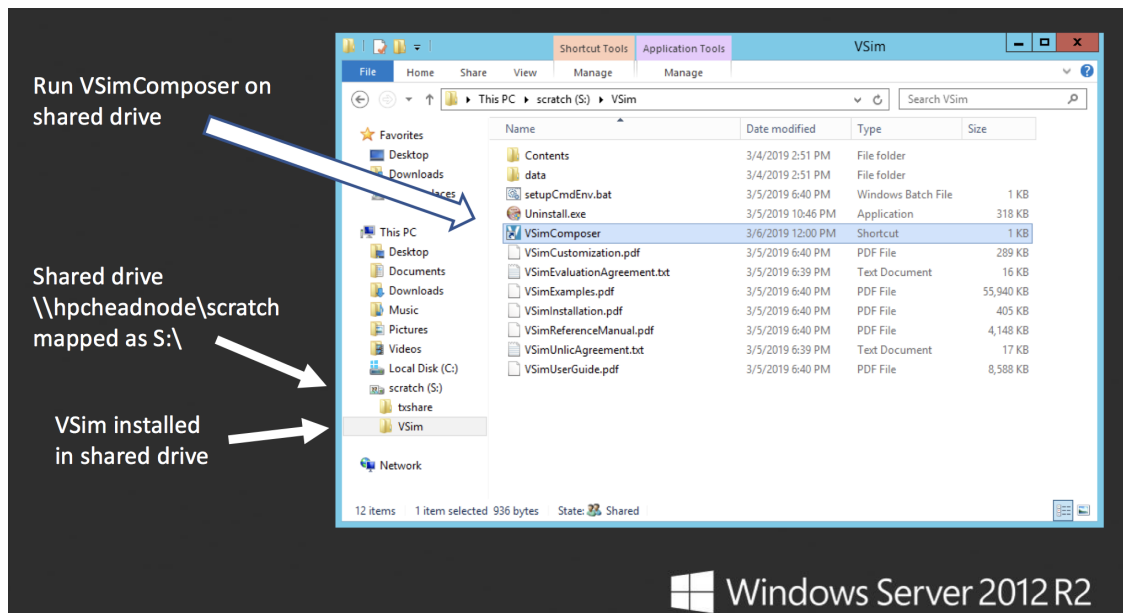
Fig. 8.11: VSim must be installed in the shared drive to run Vorpal on a Windows Cluster.
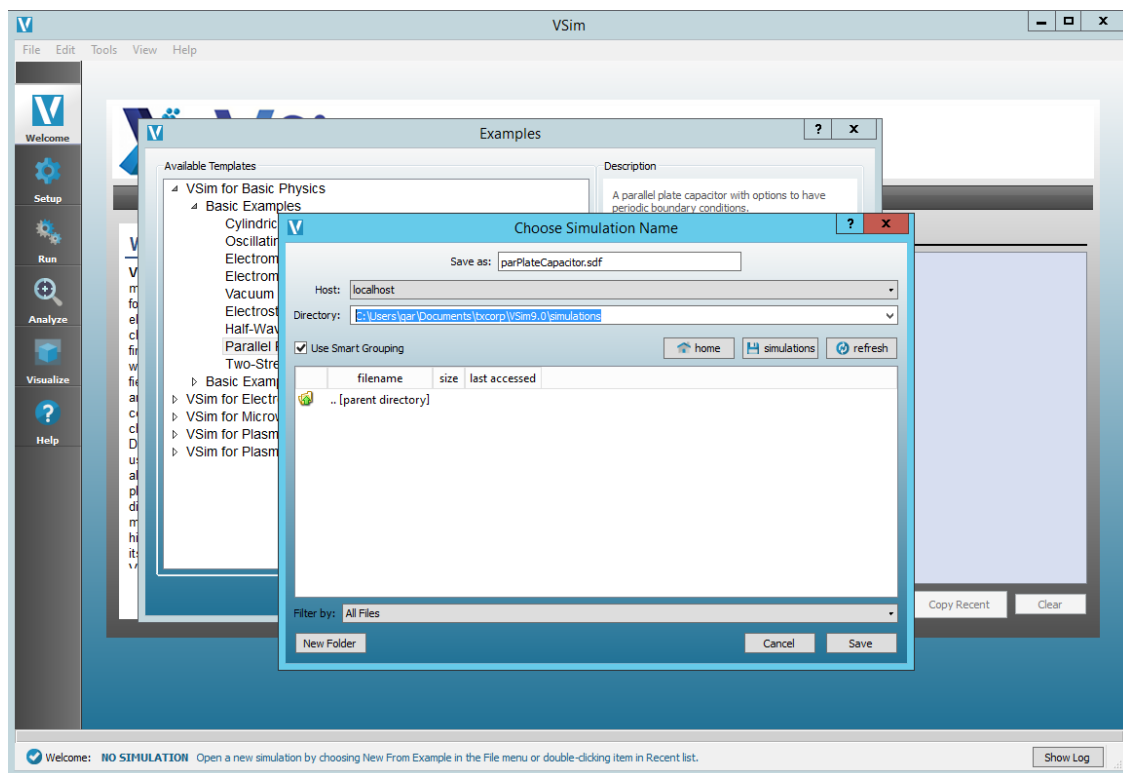
Fig. 8.12: Choose the shared drive in the Directory field so that the subsequent New Folder is on that drive.
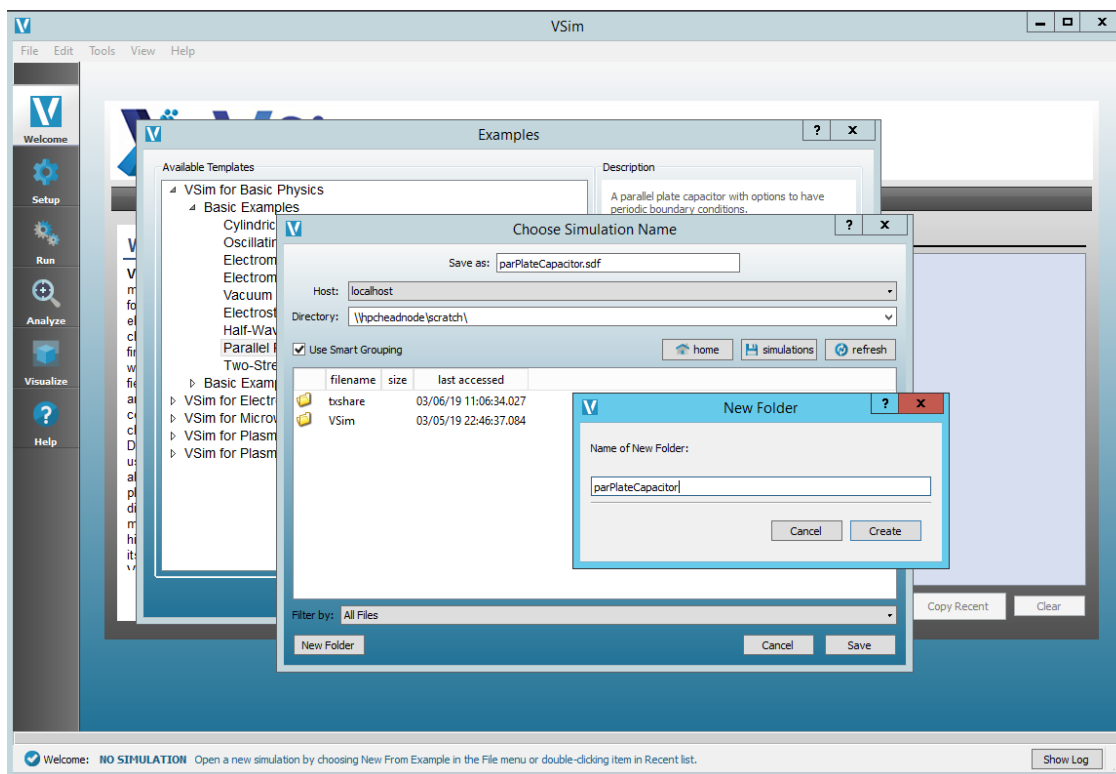
Fig. 8.13: Choose the name of the containing directory of your new simulation.

You should now have a simulation directory in your share drive and be able to see it in Windows Explorer. See Fig. 8.14
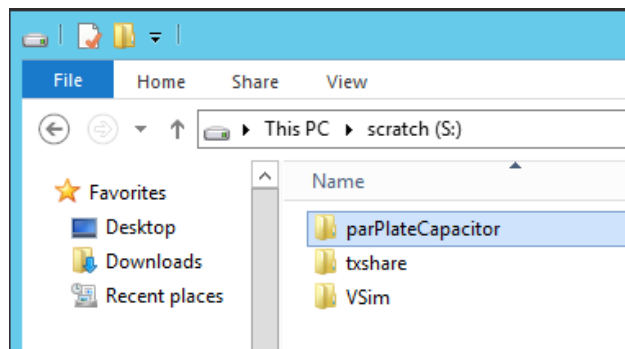
Fig. 8.14: The simulation directory needs to be on the shared drive to run a simulation on a Windows Cluster.

Create a New Cluster Job

Jobs can be started from the command line and from the Cluster Manager tool. We will show how to create and submit a job from the Cluster Manager interface and leave it to the user to follow the Microsoft documentation on how to save the Job XML and run subsequent jobs from the command line.

Start the Cluster Manager tool usually located in

C:Program FilesMicrosoft HPC Pack 2012BinHpcClusterManager.exe

This should present with a window as shown in Fig. 8.15
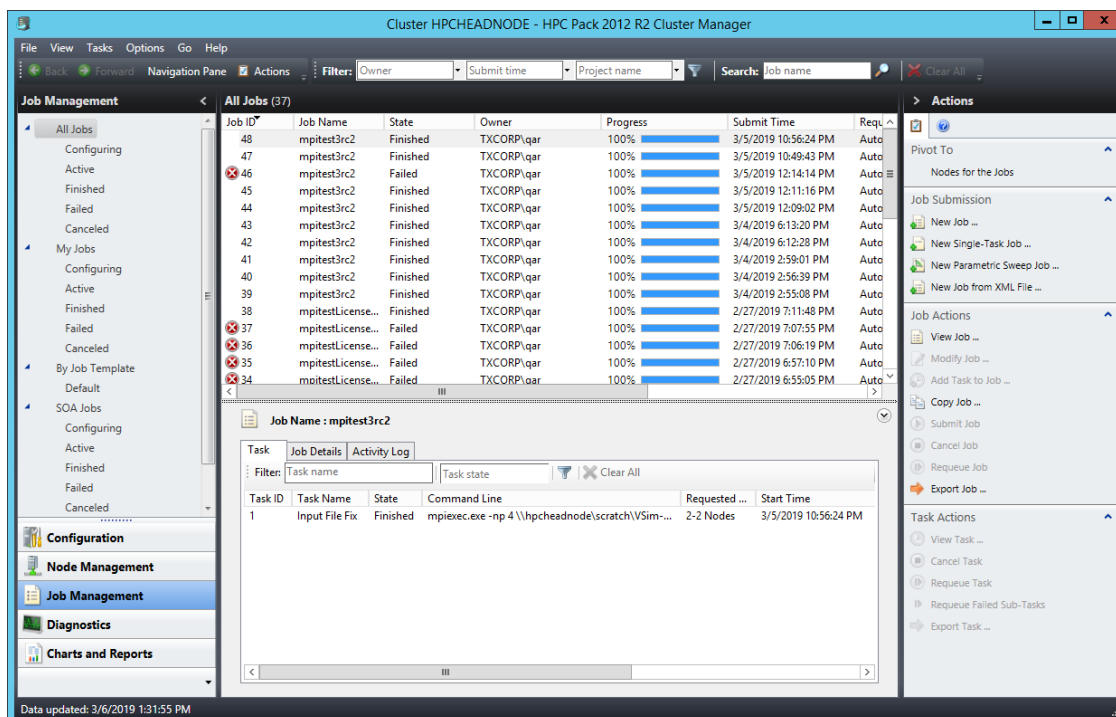
Fig. 8.15: The Microsoft HPC Pack Cluster Manager tool used to create and submit jobs to the cluster.

In the Cluster Manager, complete the following steps:

- Select the *Job Management* tab in the lower left part of the window.

- In the right *Actions* pane click *New job. . . .* This will bring up a New Job dialog

- In the *New Job* dialog, type a name in the *Job name* field as shown in Fig. 8.16

Continue with:

- Select the *Edit Tasks* item in the left menu.

- Click the Add button as shown in Fig. 8.17

Edit the task details by:

- Edit the *Task name* field if desired.

- Enter the command for running vorpal via MPI into the *Command line* field. Here you need the full path. to the vorpal executable and input file for your simulation. All the normal vorpal command-line arguments apply.

- The *Working directory* field is required to be the simulation directory that we set up above.

- The *Standard output* and *Standard error* fields are optional, but make it handy to organize the output.

- Finally, select the number of nodes you would like to run on. The "-np 4" argument along with the Minimum 2 value says that we would like to run 4 MPI processes across 2 nodes.

The job Task Details dialog is shown in Fig. 8.18

Once the job task details are set, we need to do one last set of steps:

- Select the task defined above.

Fig. 8.16: The Cluster Manager New Job dialog is a wizard to help create and submit a job.
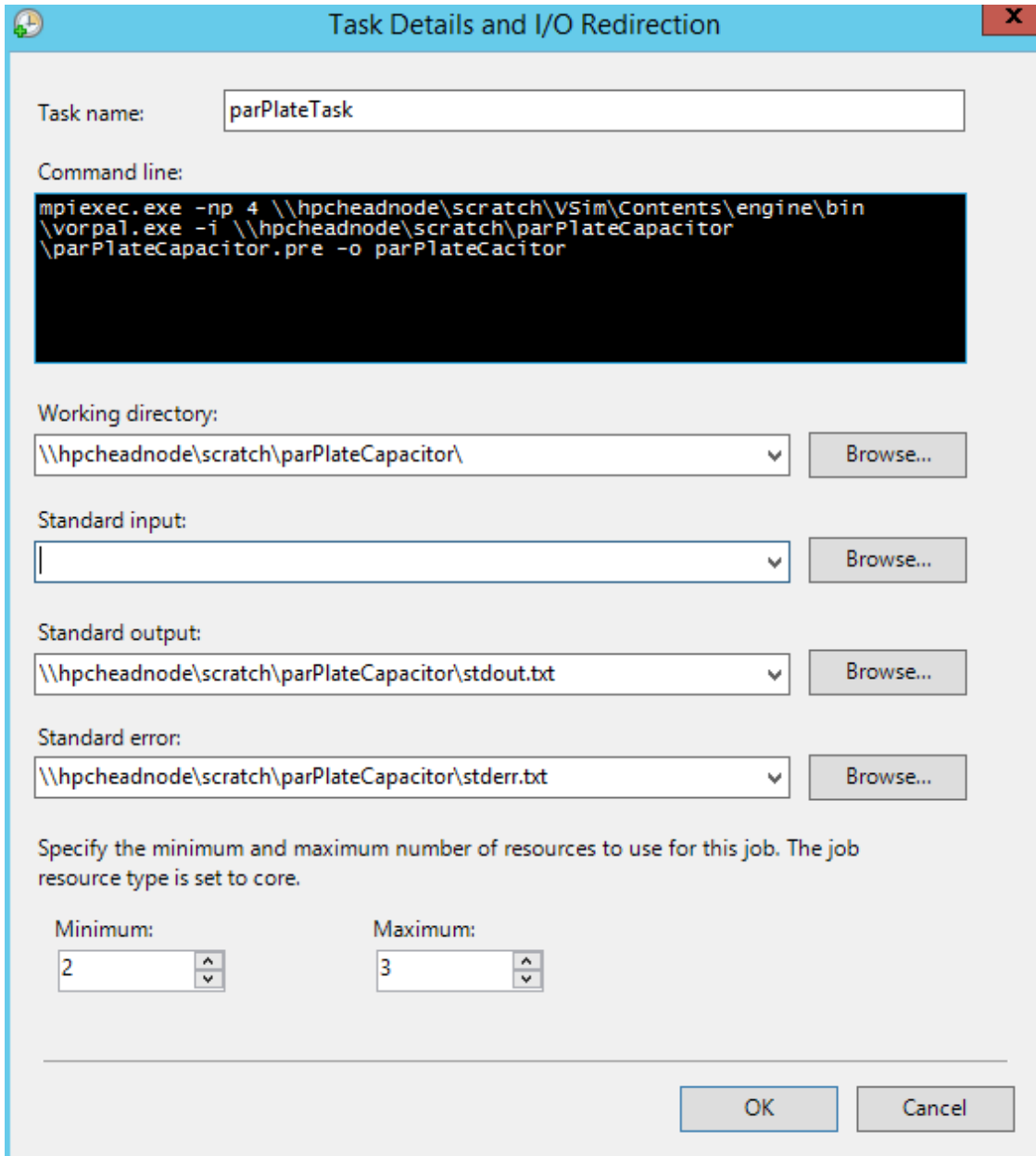
Fig. 8.17: Adding a task in the Cluster Manager New Job dialog.

Fig. 8.18: Creating parameters for the task in the Cluster Manager New Job dialog.

- Click *Environment Variables* in the *Task Properites* list. Then click the . . . button to the right of the line. The is highlighted in Fig. 8.19.
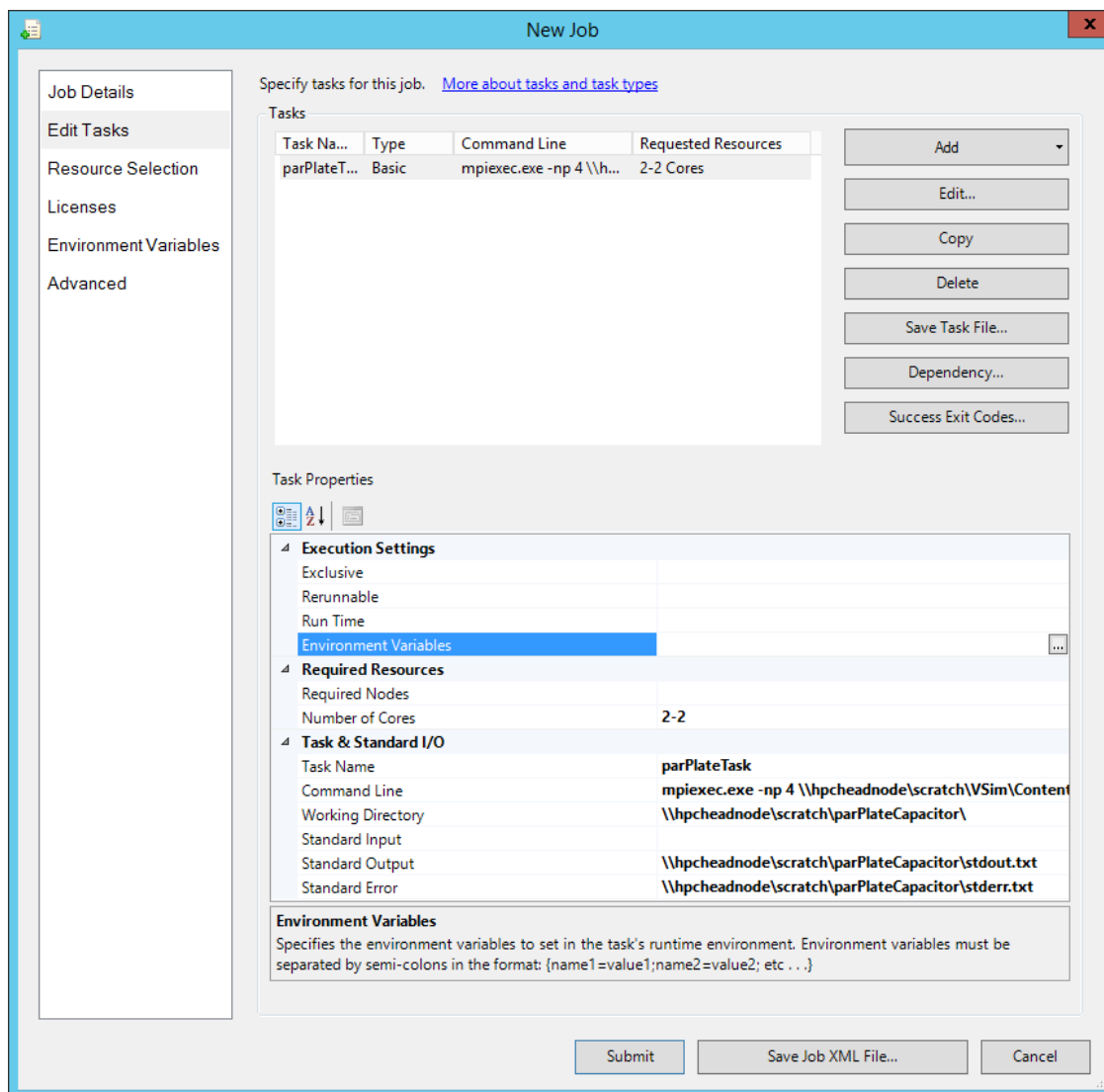
Fig. 8.19: Editing Job Properies for the task in the Cluster Manager New Job dialog.

In the Environment Variables dialog (Fig. 8.20),

click the *Add* button. This will bring up the *Add Environment Variable* dialog (see Fig. 8.21).

In this dialog, enter PYTHONPATH for the *Name* and the following for *Value*: \\hpcheadnode\scratch\VSim\Contents\engine\share\scripts; \\hpcheadnode\scratch\VSim\Contents\engine\lib\site-packages; \\hpcheadnode\scratch\VSim\Contents\engine\lib;\\hpcheadnode\scratch\VSim\Contents\bin\lib` \\hpcheadnode\scratch\VSim\Contents\bin\lib\python\lib;\\hpcheadnode\scratch\VSim\Contents` \\hpcheadnode\scratch\VSim\Contents\bin;.

You will want to replace the value of the UNC Share Path (\\hpcheadnode\scratch) with your own UNC Share Path in the PYTHONPATH line. The PYTHONPATH variable is required for all vorpal simulations to run, but other variables such as SIM_DATA_PATH may be needed for a few select simulations.
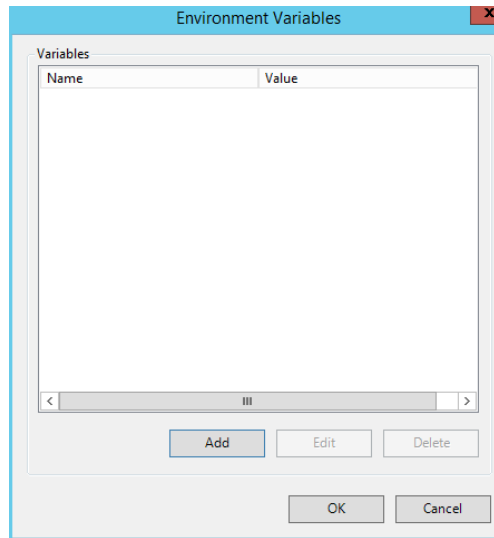
Fig. 8.20: Environment Variables dialog for the task allows user to see and add variables.
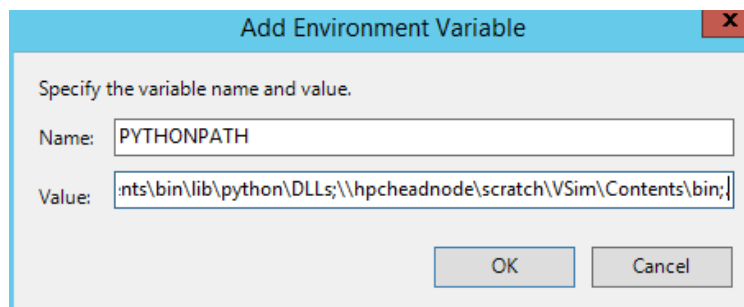
Fig. 8.21: Add Environment Variables dialog for the task to add PYTHONPATH.

8.3.3 Running Vorpal in the Cloud

Note: You will need to setup a separate cloud account in order to run Vorpal in the Cloud. These instruction are for using the Amazon Web Services cloud provider, which has nice tools for setting up an HPC Cluster.

Running Vorpal in the Amazon cloud or AWS (Amazon Web Services) on a cluster is now possible. These instructions will show how to setup a user account with the correct permissions, create a storage area, download the ParallelCluster tools, and configure and start a cluster.

These instructions are focused on the minimum needed to get the user up and running on AWS. For more detailed instructions about HPC cluster please see:

- The AWS High Performance Computing page.
- The ParallelCluster Documentation page.

The Basic Structure of an HPC AWS Cluster

In the cloud, operations are done with Web Services, so starting a cluster involves interating with server AWS services. The full architecture diagram is shown in figure Fig. 8.22 below. In these instructions we will be referring to various services and you will need to interact with the website to use these service to setup the cluster.
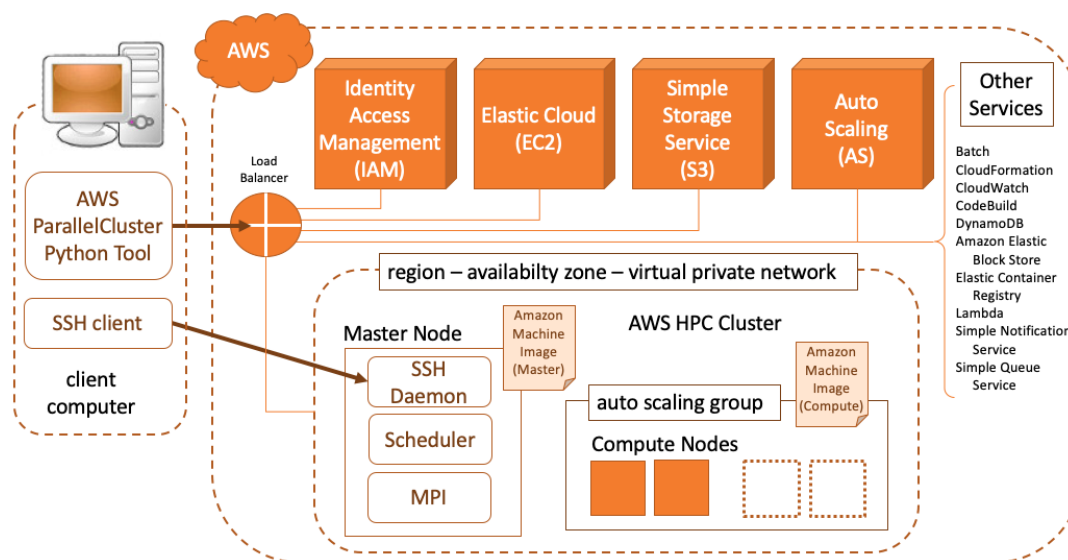
Fig. 8.22: The Amazon AWS approach in constructing an HPC Cluster using Web Services and Machine Images that have pre-installed software and networking services that automatically scale the number of compute instances to the job scheduled on the master node.

Creating AWS Accounts

Before one can run Vorpal on the AWS cloud, at least two accounts have to be setup. The root account manages information for the organization (sets up credit card payments, adds other users, etc.), so this is the first account. The second account will be for the user that runs Vorpal and will need special permissions in order to create a cluster.

Creating a Root Account

To create a root account, follow these steps:

- Goto the AWS homepage https://aws.amazon.com/

- Click the *Create a Free Account* button.

- Enter your account information, and then choose *Continue*.

- Enter your Contact Information (company or personal).

- Read and accept the AWS Customer Agreement.

- Choose *Create Account and Continue*.

- Enter your credit card information.

- Confirm your identity for example with a text.

- On the next screen, choose a plan (The basic will work).

- The final screen will ask for your business role and interests.

Creating a User Account

We need to create two policies, two groups associated with these polices, and the a user that is a member of both groups. To do this, follow these steps:

- Goto the AWS homepage https://aws.amazon.com/

- Choose *Sign in to the Console* (upper right button)

- Enter your root password

- At this point, it is good to check your default region, which is shown in the top bar at the right. You will need use the code for this region to complete the configuration below. A table of Region Names and Region Codes is given at List of Regions Codes.

Now we create two policies:

- Type IAM in the search box and then click on the link to the IAM Console

- Note the AWS Account # at the top left of the page, you will need this later.

- Click on the Policies item in the left navigation bar

- Click the *Create policy* button, which will take you to the *Create policy* editor page.

- In the *Create policy* page, click the *JSON* tab and cut-n-paste the JSON code below, replacing the <ACCOUNT_NUMBER> with the AWS account number noted above in two places and the <REGION> with your region in one place.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Action": [
              "ec2:DescribeKeyPairs",
              "ec2:DescribeRegions",
              "ec2:DescribeVpcs",
              "ec2:DescribeSubnets",
              "ec2:DescribeSecurityGroups",
```

```
            "ec2:DescribePlacementGroups",
            "ec2:DescribeImages",
            "ec2:DescribeInstances",
            "ec2:DescribeInstanceStatus",
            "ec2:DescribeSnapshots",
            "ec2:DescribeVolumes",
            "ec2:DescribeVpcAttribute",
            "ec2:DescribeAddresses",
            "ec2:CreateTags",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeAvailabilityZones"
        ],
        "Resource": "*",
        "Effect": "Allow",
        "Sid": "EC2Describe"
    },
    {
        "Action": [
            "ec2:CreateVolume",
            "ec2:RunInstances",
            "ec2:AllocateAddress",
            "ec2:AssociateAddress",
            "ec2:AttachNetworkInterface",
            "ec2:AuthorizeSecurityGroupEgress",
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:CreateNetworkInterface",
            "ec2:CreateSecurityGroup",
            "ec2:ModifyVolumeAttribute",
            "ec2:ModifyNetworkInterfaceAttribute",
            "ec2:DeleteNetworkInterface",
            "ec2:DeleteVolume",
            "ec2:TerminateInstances",
            "ec2:DeleteSecurityGroup",
            "ec2:DisassociateAddress",
            "ec2:RevokeSecurityGroupIngress",
            "ec2:RevokeSecurityGroupEgress",
            "ec2:ReleaseAddress",
            "ec2:CreatePlacementGroup",
            "ec2:DeletePlacementGroup"
        ],
        "Resource": "*",
        "Effect": "Allow",
        "Sid": "EC2Modify"
    },
    {
        "Action": [
            "autoscaling:DescribeAutoScalingGroups",
            "autoscaling:DescribeAutoScalingInstances"
        ],
        "Resource": "*",
        "Effect": "Allow",
        "Sid": "AutoScalingDescribe"
    },
    {
        "Action": [
            "autoscaling:CreateAutoScalingGroup",
            "ec2:CreateLaunchTemplate",
```

```
                "ec2:ModifyLaunchTemplate",
                "ec2:DeleteLaunchTemplate",
                "ec2:DescribeLaunchTemplates",
                "ec2:DescribeLaunchTemplateVersions",
                "autoscaling:PutNotificationConfiguration",
                "autoscaling:UpdateAutoScalingGroup",
                "autoscaling:PutScalingPolicy",
                "autoscaling:DescribeScalingActivities",
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:DeletePolicy",
                "autoscaling:DisableMetricsCollection",
                "autoscaling:EnableMetricsCollection"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "AutoScalingModify"
        },
        {
            "Action": [
                "dynamodb:DescribeTable"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "DynamoDBDescribe"
        },
        {
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DeleteTable",
                "dynamodb:TagResource"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "DynamoDBModify"
        },
        {
            "Action": [
                "sqs:GetQueueAttributes"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "SQSDescribe"
        },
        {
            "Action": [
                "sqs:CreateQueue",
                "sqs:SetQueueAttributes",
                "sqs:DeleteQueue",
                "sqs:TagQueue"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "SQSModify"
        },
        {
            "Action": [
                "sns:ListTopics",
```

8.3. Running Vorpal on a Cluster using a Queuing System 129

```
                "sns:GetTopicAttributes"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "SNSDescribe"
        },
        {
            "Action": [
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:DeleteTopic"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "SNSModify"
        },
        {
            "Action": [
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeStackResource",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStacks",
                "cloudformation:ListStacks",
                "cloudformation:GetTemplate"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "CloudFormationDescribe"
        },
        {
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:UpdateStack"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "CloudFormationModify"
        },
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Resource": "arn:aws:s3:::<REGIION>-aws-parallelcluster*",
            "Effect": "Allow",
            "Sid": "S3ParallelClusterReadWrite"
        },
        {
            "Action": [
                "iam:PassRole",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:TagRole",
                "iam:SimulatePrincipalPolicy"
            ],
```

```
                "Resource": "arn:aws:iam::<ACCOUNT_NUMBER>:role/vsim_cluster_role",
                "Effect": "Allow",
                "Sid": "IAMModify"
            },
            {
                "Action": [
                    "iam:CreateInstanceProfile",
                    "iam:DeleteInstanceProfile"
                ],
                "Resource": "arn:aws:iam::<ACCOUNT_NUMBER>:instance-profile/*",
                "Effect": "Allow",
                "Sid": "IAMCreateInstanceProfile"
            },
            {
                "Action": [
                    "iam:AddRoleToInstanceProfile",
                    "iam:RemoveRoleFromInstanceProfile",
                    "iam:GetRolePolicy",
                    "iam:GetPolicy",
                    "iam:AttachRolePolicy",
                    "iam:DetachRolePolicy",
                    "iam:PutRolePolicy",
                    "iam:DeleteRolePolicy"
                ],
                "Resource": "*",
                "Effect": "Allow",
                "Sid": "IAMInstanceProfile"
            },
            {
                "Action": [
                    "efs:DescribeMountTargets",
                    "efs:DescribeMountTargetSecurityGroups",
                    "ec2:DescribeNetworkInterfaceAttribute"
                ],
                "Resource": "*",
                "Effect": "Allow",
                "Sid": "EFSDescribe"
            },
            {
                "Action": [
                    "ssm:GetParametersByPath"
                ],
                "Resource": "*",
                "Effect": "Allow",
                "Sid": "SSMDescribe"
            }
        ]
}
```

- Click the *Review Policy* button, which takes you to the review step.

- If there are no issues with the JSON, then enter the name *vsim_user_policy*, otherwise, click the *Previous* button and and fix any issues. Once you are ready to create the policy, click *Create policy* to complete the first policy.

- Back in *Policies* page click the *Create policy* button for a second policy.

- In the *Create policy* page, click the *JSON* tab and cut-n-paste the JSON code below, replacing the four <ACCOUNT_NUMBER> with the AWS account number noted above and replacing the five <REGION> with your

region.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Action": [
              "ec2:DescribeVolumes",
              "ec2:AttachVolume",
              "ec2:DescribeInstanceAttribute",
              "ec2:DescribeInstanceStatus",
              "ec2:DescribeInstances",
              "ec2:DescribeRegions"
          ],
          "Resource": [
              "*"
          ],
          "Effect": "Allow",
          "Sid": "EC2"
      },
      {
          "Action": [
              "dynamodb:ListTables"
          ],
          "Resource": [
              "*"
          ],
          "Effect": "Allow",
          "Sid": "DynamoDBList"
      },
      {
          "Action": [
              "sqs:SendMessage",
              "sqs:ReceiveMessage",
              "sqs:ChangeMessageVisibility",
              "sqs:DeleteMessage",
              "sqs:GetQueueUrl"
          ],
          "Resource": [
              "arn:aws:sqs:<REGION>:<ACCOUNT_NUMBER>:parallelcluster-*"
          ],
          "Effect": "Allow",
          "Sid": "SQSQueue"
      },
      {
          "Action": [
              "autoscaling:DescribeAutoScalingGroups",
              "autoscaling:TerminateInstanceInAutoScalingGroup",
              "autoscaling:SetDesiredCapacity",
              "autoScaling:UpdateAutoScalingGroup",
              "autoscaling:DescribeTags",
              "autoScaling:SetInstanceHealth"
          ],
          "Resource": [
              "*"
          ],
          "Effect": "Allow",
          "Sid": "Autoscaling"
```

```
    },
    {
        "Action": [
            "cloudformation:DescribeStacks",
            "cloudformation:DescribeStackResource"
        ],
        "Resource": [
            "arn:aws:cloudformation:<REGION>:<ACCOUNT_NUMBER>:stack/parallelcluster-
↪*/*"
        ],
        "Effect": "Allow",
        "Sid": "CloudFormation"
    },
    {
        "Action": [
            "dynamodb:PutItem",
            "dynamodb:Query",
            "dynamodb:GetItem",
            "dynamodb:DeleteItem",
            "dynamodb:DescribeTable"
        ],
        "Resource": [
            "arn:aws:dynamodb:<REGION>:<ACCOUNT_NUMBER>:table/parallelcluster-*"
        ],
        "Effect": "Allow",
        "Sid": "DynamoDBTable"
    },
    {
        "Action": [
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::<REGION>-aws-parallelcluster/*"
        ],
        "Effect": "Allow",
        "Sid": "S3GetObj"
    },
    {
        "Action": [
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::<REGION>-aws-parallelcluster/*"
        ],
        "Effect": "Allow",
        "Sid": "S3PutObj"
    },
    {
        "Action": [
            "sqs:ListQueues"
        ],
        "Resource": [
            "*"
        ],
        "Effect": "Allow",
        "Sid": "SQSList"
    },
```

8.3. Running Vorpal on a Cluster using a Queuing System

```
    {
        "Action": [
            "iam:PassRole",
            "iam:CreateRole"
        ],
        "Resource": [
            "arn:aws:iam::<ACCOUNT_NUMBER>:role/parallelcluster-*"
        ],
        "Effect": "Allow",
        "Sid": "BatchJobPassRole"
    }
  ]
}
```

- Click the *Review Policy* button, which takes you to the review step.

- If there are no issues with the JSON, then enter the name *vsim_creator_policy*, otherwise, click the *Previous* button and and fix any issues. Once you are ready to create the policy, click *Create policy* to complete the first policy.

Now we create two groups for the two policies:

- Click on the Groups item in the left navigation bar

- Click *Create New Group*, enter the name vsim_user_group, and then click the *Next Step* button.

- In the *Attach Policy* page, enter vsim in the *Policy Type* search box, then check the checkbox by vsim_user_policy, and click the *Next Step* button.

- In the *Review* page, click the *Create Group* button

- Click the *Create New Group* button a second time, but this time name the group vsim_creator_group and attach the vsim_creator_policy policy.

Next, we add a Role for the EC2 service to use to be able to access the AWS services:

- Click on the Roles item in the left navigation bar

- Click *Create role*, choose the EC2 Service to use the role. Click the *Next: Permissions* button.

- Enter vsim in the search box to filter the policies to attach and select both the vsim_creator_policy and the vsim_user_policy, then click *Next: Tags*.

- No tags are necessary, so if you don't want any, click *Next: Review*.

- Enter the Role name, vsim_cluster_role and click *Create role*

- Create one more role. Click *Create role*, choose the EC2 Auto Scaling as the service and then chose EC2 Auto Scaling as the use case.

- Click *Next: Permissions*, *Next: Tags*, *Next: Review*, and then *Create role*.

We then add the user:

- Click the Users item in the left navigation bar

- Click the *Add User* button

- Enter a user name and select the *Programmatic access* option and then click the *Next:Permissions* button.

- In the *Add user to group* section, check both the vsim_users_group and the vsim_creator_group checkboxes. Then click the *Next: Tags* button.

- No tags are required, so you can move on to click the *Next: Review* button and if all is correct, then click the *Create user* button.

- The next page should have a green box with the word *Success* in it. Keep this page open because it is the only time you will get access to the secret key for the user. And you will want to make a file on your local computer with this key at this time. To do this, make a file *~/.aws/config* with the following:

```
$ mkdir ~/.aws
$ touch ~/.aws/config
$ chmod 600 ~/.aws/config
```

Then edit this "config" file. Copy the following contents into the config file and replace the <ACCESS_KEY_ID> and the <SECRET_ACCESS_KEY> from the browser and <REGION> from the region you are using.

```
[default]
aws_access_key_id=<ACCESS_KEY_ID>
aws_secret_access_key=<SECRET_ACCESS_KEY>
region=<REGION>
```

This file will be used later to contact the various Amazon Web Services needed to create a cluster.

You can also download a comma-separated-values (.csv) file with the user's creditials to save the access key information.

Create a Long Term Storage Space

These instructions take you through setting up an S3 storage bucket and uploading VSim to it.

- Go to the S3 dashboard, by choosing Services->Storage->S3

- Click the *Create Bucket* button

- Enter unique name for the bucket and note it for later as <BUCKET_NAME> and ensure that the region is the same as <REGION>, then click the *Create* button in the lower left (not the *Next* button in the lower right).

- In the list of buckets, click on the link to your bucket.

- Click the *Upload* button and then the *Add files* button. Navigate to where your VSim Linux installer (VSim-11.0.0-Linux64.tar.gz) is located. Then click the *Add more files* link and this time navigate to the license file you have been given.

- Click *Next*. Ensure that your AWS user account is the owner and has full permissions to Read and Write. Click *Next*.

- Choose *Standard* for the Storage Class unless you know why would otherwise. Click *Next*.

- Review the selections and click *Upload* if ready.

Create a Short Term (mountable) Storage System

Short term storage is only used during the cluster instance operation. Files are sync'd back and forth to the long term storage on cluster creation and deletion. There are two basic options for this storage: (1) EFS, an NFS type system; and (2) FSx, a Lustre high performance system. Option 2 is more expensive, but more performant and you can create both and go back and forth between clusters with one or the other. So, we give instructions for using EFS first.

- Go to the EFS dashboard, by choose Services->Storage->EFS

- Ensure you have the region choice in the top bar to be your region.

- Click the *Create file system* button

- The next screen is a review of the default settings. These will be fine unless you know that you want something different. So, click the *Next Step* button.

- In the *Configure optional settings* page, scroll down to the *Choose performance mode* section and check *Max I/O* option, which will be needed for clusters with many compute nodes. Click the *Next Step* button.

- In the *Review and create* page, you will see the Virtual Private Cloud (vpc) that the file system is within and this includes a set of Availability Zones (subnets) within your region. This is good to be aware of and these IDs will be used later in configuring your cluster.

- When you are done reviewing, click *Create File System*.

Create key pair to use with SSH

- Go to the EC2 dashboard, by choosing Services->Compute->EC2

- Click on the *Key Pairs* link in top-middle of the page.

At this point we need to upload a public key. Usually, this is in the ~/.ssh directory. On Mac, one can not navigate to the directory via the web browser, we can copy the public key to a directory that is available, like *Documents*.

```
$ cp ~/.ssh/id_rsa.pub ~/Documents
```

- Click the *Import Key Pair* button name.

- Next to the *Load public key from file* option, click the *Choose File* button and then find the public key in the Documents directory.

- Change the name to *vsimcluster*.

- Click the *Import* button.

Downloading the ParallelCluster tools

These instructions describe how to download a Python 3 based tool from AWS needed to lauch a cluster from the command-line:

Download and install Python 3 if you don't already have it. On Mac, this can be done with homebrew:

```
$ python --version
$ brew install python3   # if not version 3.7
```

Ensure that python3 is first in your PATH then install the cluster tools. For example, on Mac you would do the following:

```
$ which pip3      # should return a full path to python3 distribution
$ pip3 install aws-parallelcluster --upgrade --user
$ export PATH=~/Library/Python/3.7/bin:$PATH
$ which pcluster     # should return a full path
```

On linux, python puts pip installed packages in ~/.local/bin and this is likely already in your path, so there would be no need to for the export line.

After pcluster is installed create the configuration directory and file (~/.parallelcluster/config) with the following contents, replacing XXXXXXXX with the vpc and subnet IDs. The vpc id can be found at the EC2 dashboard by clicking on the *Default VPC* link in the upper right of the page. And in the *VPC Dashboard*, click on the *Subnets* link in the left navigation bar. You should see a subnet ID for each Availabilty Zone. Choosing the first one is fine.

```
[aws]
aws_region_name = <REGION>

[cluster default]
key_name = vsimcluster
initial_queue_size = 0
max_queue_size = 16
min_queue_size = 0
placement_group = DYNAMIC
cluster_type = spot
base_os = centos7
scheduler = slurm
master_instance_type = c4.4xlarge
compute_instance_type = c4.4xlarge
vpc_settings = vsimvpc
ebs_settings = vsimebs
s3_read_write_resource = arn:aws:s3:::<BUCKET_NAME>*

[vpc vsimvpc]
vpc_id = vpc-<VPC_ID>
master_subnet_id = subnet-<SUBNET_ID>

[ebs vsimebs]
shared_dir = /vsim
volume_type = gp2
volume_size = 100

[global]
cluster_template = default
update_check = true

[aliases]
ssh = ssh {CFN_USER}@{MASTER_IP} {ARGS}
```

Creating a Cluster

At this point, you should be setup to start an AWS Cloud Cluster with the ParallelCluster tool.

```
$ export PATH=~/Library/Python/3.7/bin:$PATH
$ pcluster create mycluster
```

This will take several minutes. You can watch for the Master node being created under the *Running Instances* link on the EC2 Dashboard. Also, you may receive a message on your first startup saying that your "request for accessing resources in this region is being validated"

When the prompt returns, the login information will be printed out. Use that login information (account and public IP) to login, with a command like:

```
ssh ec2-user@100.20.239.156
```

You should not need a password because you will be using your private/public key pair that you imported above. Once you are log into the cluster, you can look around with commands like the following

```
$ ls /
$ which sbatch
$ sinfo
```

You should see the /vsim file system and that the scheduler is present and working. You can try a simple slurm batch script with the following contents:

```sh
#!/bin/sh
#SBATCH --job-name=test
#SBATCH --output=test.out
#SBATCH --error=test.err
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --time=00:01:00
/bin/hostname
srun -l /bin/hostname
srun -l which mpirun
```

And submit it with (Do we need the –nodelist="$SLURM_JOB_NODELIST" arg???)

```
$ sbatch test.slm
```

You can view its progress with the squeue command:

```
$ squeue
```

And you can check that compute nodes are started by going back to the EC2 Running Instances page in the browser. When the job is completed, you should see the standard output (test.out) and and standard error (test.err) files. The error file should be empty and the output file should have hostnames and paths to MPI.

Running A Job on the Cluster

If the cluster is working well, and you are confident of how to run a batch slurm job, then you can setup VSim on the cluster. We do this by sync'ing the S3 storage with the EFS share.

```
$ aws s3 sync s3://<BUCKET_NAME> /vsim
$ cd /vsim
$ tar xzvf VSim-11.0.0-Linux64.tar.gz
$ cp vsim11.txlic VSim-11.0/licenses/license.txt
```

And then to test running VSim, copy an example directory and translate the SDF file with:

```
$ cd /vsim
$ mkdir empulse/
$ cp VSim-11.0/Contents/examples/VSimBase/VisualSetup/emPulseInVacuum/* empulse/
$ source VSim-11.0/VSimComposer.sh
$ cd empulse
$ sdf2vpre -s emPulseInVacuum.sdf -p emPulseInVacuum.pre
```

Here is a slurm batch script to run a job on the cluster with this input file:

```bash
#!/bin/bash
#SBATCH --job-name=empulse
#SBATCH --output=empulse.out
#SBATCH --error=empulse.err
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --ntasks=4
#SBATCH --time=00:10:00
```

```
source /vsim/VSim-11.0/VSimComposer.sh
srun  --hint=nomultithread --ntasks-per-node=16 vorpal -i emPulseInVacuum.pre -n 100 -
↪d 50
```

If the file containing the above is named, *empulse.slm*, then this job is submitted with

```
sbatch empulse.slm
```

You can check on your job with

```
squeue
```

and you can stop the job with

```
scancel JOBID
```

where JOBID is the job id returned by squeue.

Shutting Down the Cluster

When you are through running you simulations on the cluster it should be shutdown with the following command:

```
$ export PATH=~/Library/Python/3.7/bin:$PATH
$ pcluster delete mycluster
```

OUTPUT DATA

9.1 HDF5 Format Data Output Files

Vorpal outputs data in two forms, text and HDF5. Text output is used for progress reporting, while HDF5 is used for data files. The HDF5 data files have the `.h5` suffix.

Hierarchical Data Format Version 5 (HDF5) is a library and file format for storing graphical and numerical data and for transferring that data between computers. Vorpal and VSimComposer output data in HDF5 format. The Hierarchical Data Format was developed by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign. For more information about HDF5 (See http://hdfgroup.org/HDF5).

9.2 Dumping Fields, Particles, and GridBoundaries

The following are the general dumping options that can be set in field, particle, and grid boundary blocks. These can be used when the user wishes to customize the dumping options for the particular blocks and not change the global dumping options.

- dumpOncePerRun (boolean)

  If true, this object will be dumped only once per run. This specification can co-exist with other specifications.

- commandLineDumpPeriodicity (integer)

  This option may be automatically set by vorpal to override other attributes specifying when this object should dump its data.

- dumpPeriodicity (integer)

  If dumpPeriodicity = p, then this object will be dumped to disc when its timestep mod p equals 0.

  If p=0, this object " "will never be dumped.

- dumpPeriod (integer)

  (deprecated: please use dumpPeriodicity instead)

  If dumpPeriodicity = p, then this object will be dumped to disc when its timestep mod p equals 0.

  If p=0, this object will never be dumped.

- dumpSteps (integer list)

  A list of the time steps at which this object should dump its data to disc.

- dumpSteps (expression)

A function of a scalar integer n (the current time step) that returns 1 if this object should be dumped at step n, and 0 if not.

To use an Expression, it must be specified within the <Expression dumpSteps> block:

```
expression = (a function of 'n', the timestep, that yields 0 or 1)
For example, expression = ( mod(n, 100) == 0 )
or           expression = ( or(n==100, n==200))
```

Vorpal produces one HDF5 file for each field or species at each dump time. For example, if the simulation parameter `nsteps = 100` and the simulation parameter `dumpPeriodicity = 10`, Vorpal dumps data 10 times during the simulation and outputs a total of 10 HDF5 files for each field or species while running the simulation. Vorpal also produces one `Globals` file at each dump containing data that is general to the whole simulation, as opposed to one species or field. Finally, Vorpal puts out a `History` file containing the data of the specified histories.

9.3 Change the Names of Output Files

If you want to change the names of the output files, which include the .h5 files, you can specify the -o output option when you run Vorpal.

For example, you want to replace `emPlaneWave` with `emPlaneWaveTest1` in the names of the `emPlaneWave` simulation's output files. Run Vorpal from the command line using this command:

```
vorpal -i emPlaneWave.in -o emPlaneWaveTest1
```

The output files will be:

- `emPlaneWaveTest1_all_1.txt`
- `emPlaneWaveTest1_comms_0.txt`
- `emPlaneWaveTest1_completed.txt`
- `emPlaneWaveTest1_dumpedobjs_0.txt`
- `emPlaneWaveTest1_electrons_1.h5`
- `emPlaneWaveTest1_Globals_1.h5`
- `emPlaneWaveTest1_SumRhoJ_1.h5`
- `emPlaneWaveTest1_YeeStaticElecField_1.h5`

9.4 Displaying the Content of .h5 Files

The **h5dump** utility converts the binary data in `.h5` files into human-readable ASCII data in `.txt` files, and is available for all the platforms on which Vorpal runs. You can download the utility from the HDF5 website (http://hdfgroup.org/HDF5).

The basic command is:

```
h5dump -o output_file_name.txt your_h5_file.h5
```

So, to convert the `emPlaneWave_electrons_1.h5` to text format, use this command:

```
h5dump -o emPlaneWave_electrons_1.txt emPlaneWave_electrons_1.h5
```

9.5 General Structure of Simulation Output .h5 Files

For each type of output file below, main data entries within that output file are displayed as a list of fields at the same level within the list. For those data fields within an output file that contain one or more subcategories of data, subcategories appear in an indented list below the main data category to which the subcategories apply.

Type of Output File: Globals

```
compGridGlobal
runInfo
time
```

Type of Output File: SumRhoJ

```
SumRhoJ
    int array [NX NY NZ 4]
    SumRhoJ[0] = Rho, charge density
    SumRhoJ[1] = Jx
    SumRhoJ[2] = Jy
    SumRhoJ[3] = Jz
compGridGlobal
compGridGlobalLimits
runInfo
time
```

Type of Output File: Fluid

```
NeutralGasName
    int array [NX NY NZ 1]
    NeutralGasName[0] = Density
compGridGlobal
compGridGlobalLimits
runInfo
time
```

Type of Output Files: emMultiField – MagMultiField

```
MagMultiField
    int array [NX NY NZ 3]
    MagMultiField[0] = Bx
    MagMultiField[1] = By
    MagMultiField[2] = Bz
compGridGlobal
compGridGlobalLimits
derivedVariables
runInfo
time
```

Type of Output File: GridBoundary name

```
name
    int array [NX NY NZ 2]
    name[0] = true (1) or false (0) is the lower left front corner inside or not?
    name[1] = true (1) or false (0) is the cell center inside or not?
nameLargeBndryFaces
nameLargeFaceFracs
nameSmallBndryFaces
```

```
nameSmallFaceFracs
nameStairStepBndryEdgesData
nameStairStepBndryFacesData
compGridGlobal
compGridGlobalLimits
derivedVariables
poly
runInfo
time
```

Type of Output File: History

```
runInfo
historyName1
historyName2
```

Note: These `historyName` arrays contain time data pertaining to the type of history chosen in the input file.

Type of Output File: species

```
species
  int array [NX NY NZ 6]
  species[0] = x position
  species[1] = y position
  species[2] = z position
  species[3] = x velocity
  species[4] = y velocity
  species[5] = z velocity
compGridGlobalLimits
runInfo
time
```

Note: The information above is representative of 3D data. The actual number of elements in an array may vary depending on the dimensionality of the simulation. The number of elements in a species output file will also vary based on the type of species used. For more information on the output of species data, see the next section.

9.6 Columns in Particle Simulation .h5 Output Files

Below is a table displaying how the columns in particle simulation `.h5` output files for various species kinds correspond to the columns that can be seen in the `.h5` file when the file is opened using a tool such as **HDFView**. HDFView may be downloaded for free from the HDF Group at http://www.hdfgroup.org/hdf-java-html/hdfview/. HDFView distributions are available for 32-bit and 64-bit Linux, Mac, and Windows platforms.

Table 9.1: **Columns in .h5 in Particle Simulation Output Files**

Species	Number of Columns	Comma-separated Columns
cmplxRelBorisDF	5 + NDIM	x,[y,[z]]Px, Py, Pz, real weight, imaginary weight
envBoris	5 + NDIM	x,[y,[z]]Px, Py, Pz, tag, weight
freeRel	3 + NDIM	x,[y,[z]]Px, Py, Pz
freeRelVW	4 + NDIM	x,[y,[z]]Px, Py, Pz, weight
noMove	3 + NDIM	x,[y,[z]]Px, Py, Pz
noMoveVW	4 + NDIM	x,[y,[z]]Px, Py, Pz, weight
nonRelBoris	3 + NDIM	x,[y,[z]]Px, Py, Pz
nonRelEs	3 + NDIM	x,[y,[z]]Px, Py, Pz
relBoris	3 + NDIM	x,[y,[z]]Px, Py, Pz
relBorisBallisticVW	4 + NDIM	x,[y,[z]]Px, Py, Pz, weight
relBorisCyl	3 + NDIM	u0,[u1,[u2,]]P0, P1, P2 (see Legend)
relBorisDF	4 + NDIM	x,[y,[z]]Px, Py, Pz, weight
relBorisEffMassExtd	2 + 3(NDIM always 3)	x, y, z, Px, Py, Pz, valley index, weight (always 1)
relBorisFuncVW	4 + NDIM	x,[y,[z]]Px, Py, Pz, weight
relBorisTagged	4 + NDIM	x,[y,[z]]Px, Py, Pz, tag
relBorisVW	4 + NDIM	x,[y,[z]]Px, Py, Pz, weight
relBorisVWTagged	5 + NDIM	x,[y,[z]]Px, Py, Pz, tag, weight
relBorisVWScale	6 + NDIM	x,[y,[z]]Px, Py, Pz, tag, scale parameter, weight
cell	3 + NDIM	x,[y,[z]]Px, Py, Pz
cell(VW)	4 + NDIM	x,[y,[z]]Px, Py, Pz, weight
cell(tagged)	4 + NDIM	x,[y,[z]]Px, Py, Pz, tag
cell(VW, tagged)	5 + NDIM	x,[y,[z]]Px, Py, Pz, tag, weight

Note: The cell species (`kind = cell`) set variable weight (VW) and tagging with the `variableWeightParticle` and `taggedParticle` parameters.

Table 9.2: **Legend**

NDIM	Number of Dimensions: 1, 2, or 3	
	Dimension Notation	
1D	2D	3D
x	x y	x y z
1D	1D,[2D]	1D,[2D,[3D]]
	Cylindrical Coordinates	
Polar	u0, u1, u2	r, phi, z
Cylindrical	u0, u1, u2	z, r, phi
Tubular	u0, u1, u2	phi, z, r
	Momentum/Mass Notation Convention	
	momentum/mass = gamma*v = P	
gamma*vx:Px	gamma*vy: Py	gamma*vz: Pz

9.7 HDFView Example Simulation .h5 Output File Illustration

Below are **HDFView** displays of Vorpal field and species `.h5` output files from an example simulation, as well as brief descriptions of said output. HDFView may be downloaded for free from the HDF Group (http://www.hdfgroup.org/hdf-java-html/hdfview/)
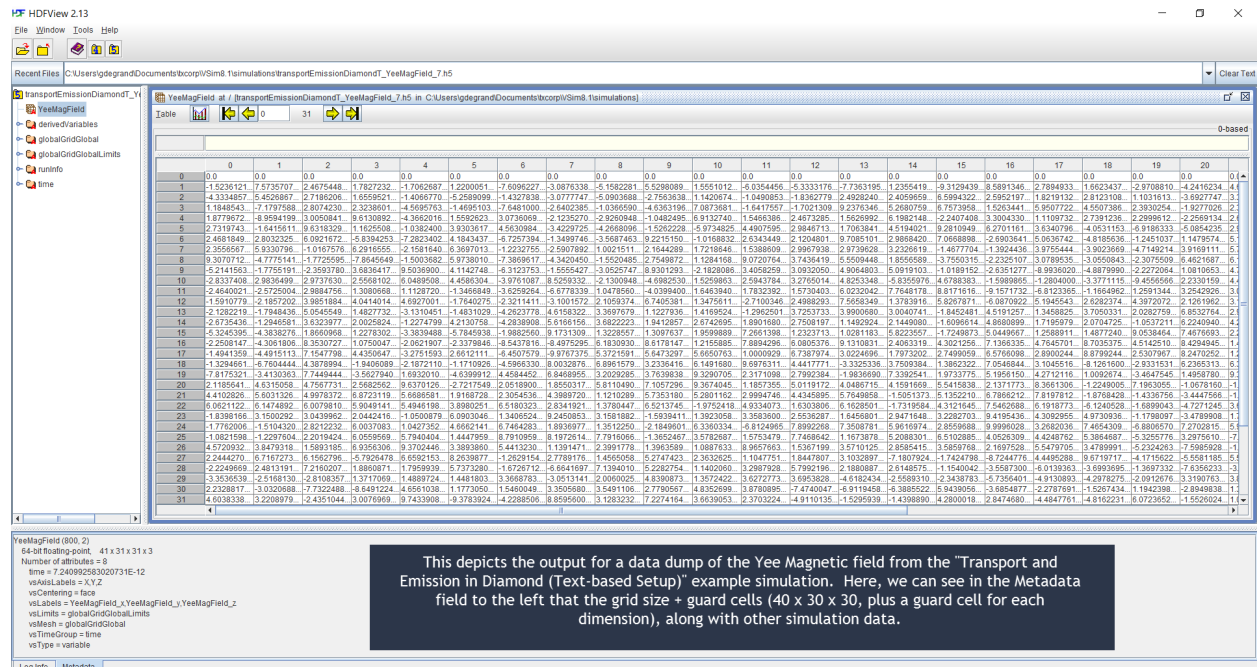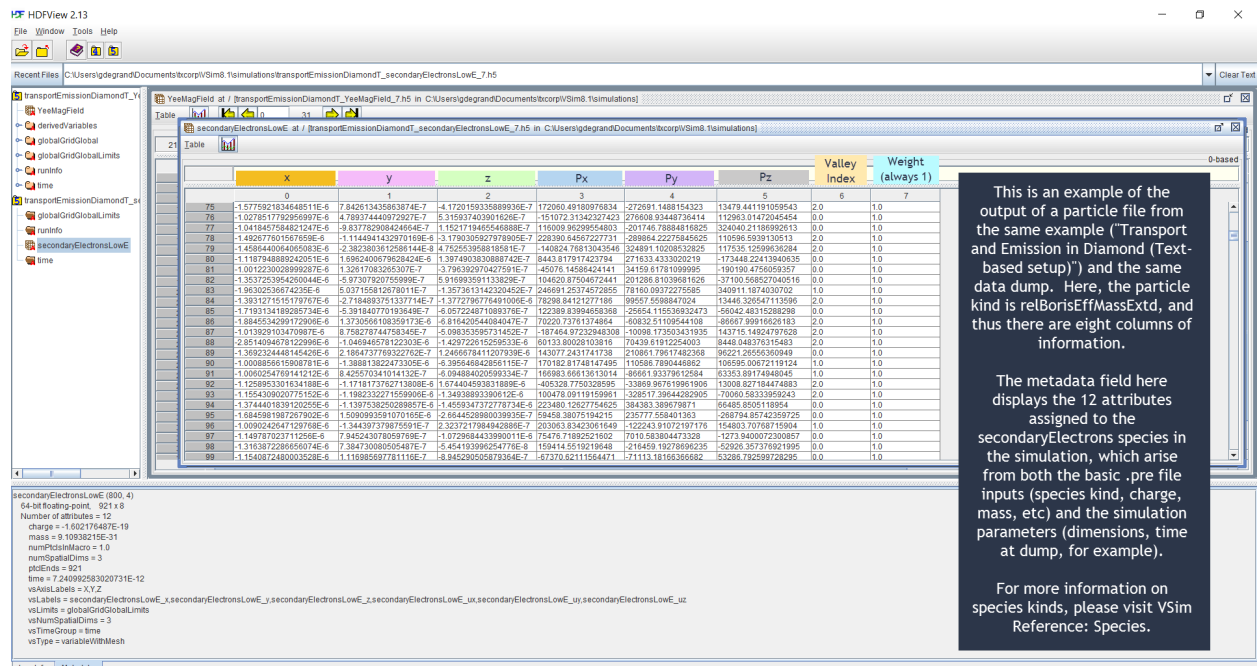
Fig. 9.1: HDFView display of a field file.

Fig. 9.2: HDFView display of particle file.

DATA ANALYSIS

10.1 Overview of Using Analyzers

It is possible to run postprocessing analyzers within the VSimComposer environment. These analyzers can process data generated in a simulation and write that data to a .h5 file that can then be visualized like any other simulation data.

Here, we will discuss the basic process of using one of the many analyzers included with VSim. It is also possible to import custom analysis scripts. For more details on these, please see VSim Customization: Create Your Own Script.

In this section, we will go through the basics of running analyzers, and will incorporate the two stream example (visual setup) to illustrate the processes.

10.1.1 Using An Example To Demostrate How to Use an Analyzer

First we open an example and run a simulation in order to get some data to analyze. Open the two stream example and follow along:

- Select the *New -> From Example* menu item in the *File* menu.

- In the resulting *Examples* window, expand the *VSim for Basic Physics* option.

- Expand the *Basic Examples* option.

- Select *Two-Stream Instability* and press the *Choose* button.

- In the resulting dialog, create a new folder if desired, and press the *Save* button to create a copy of this example in your run area.

- This will open the Two-Stream Instabilitiy example. For this exercise, we will use the default parameters.

See Fig. 10.1.

10.1.2 Select an Analyzer (Installed with VSim)

VSimComposer allows the user to select a analzer from the **Analyze** window. You can type in the name of an analyzer to filter and shorten the list or double-click any analyzer in the list. You can also select the *Import Analyzer* button at the bottom to import your own customized script.

For the two stream example, wait until the simulation run has completed and then click on the **Analyze** tab located below the **Run** tab on the left hand side of the screen.

- In the *Analysis Controls* pane, a list of all of the analyzers will be available.

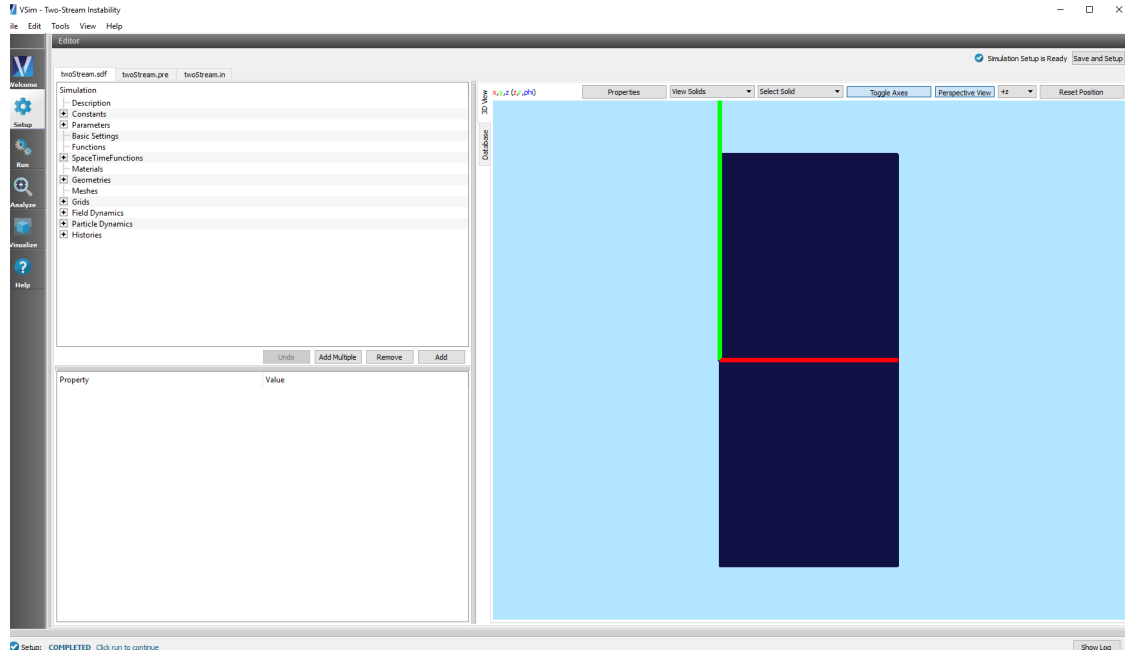- For this example, we will select *computePtclNumDensity.py*.

Fig. 10.1: Setup Window for Two Stream Example

- In the *Analysis Results* pane and under the *Outputs* tab, instructions as to which variables have to be given by the user will appear.

- For this particular analyzer, the only two variables are the *simulationName* and the *speciesName*. For *simulationName*, give it the name of the simulation, in our case *twoStream*. The *speciesName* parameter is either *electrons0* or *electrons1*, the two species of electrons in this simulation.

See Fig. 10.2.

This process should be the same for both visual and text-based simulations.

However, there are different processes for keeping analyzers available for your simulation through closing and reopening. For visual setup, simply run your simulation and analyzer of choice as usual, and then return to the **Setup** tab and select the *Save and Setup* button in the top right. This process saves your analyzer in the .sdf file for your simulation.

10.1.3 Running the Analyzer

Click the *Analyze* button located at the top right of the *Control* pane.

10.1.4 Output of an Analyzer

The data generated from the execution of an analyzer will be stored as a .vsh5 file and is visualizable underneath the **Visualize** tab. Be aware that the analyzer may also produce data that it writes to the screen or as hdf5 output.

Finishing up with the two stream example, the data we got from *computePtclNumDensity.py* can be visualized through the following:

- Open the **Visualize** tab. You may need to click on the *Reload Data* button at the top right of the *Visualization Controls* pane if you have gone back and forth between the windows and the VSim computational engine has generated more data.

- Switch the *Data View* drop-down menu at the top left of the pane to *1-D Fields*.
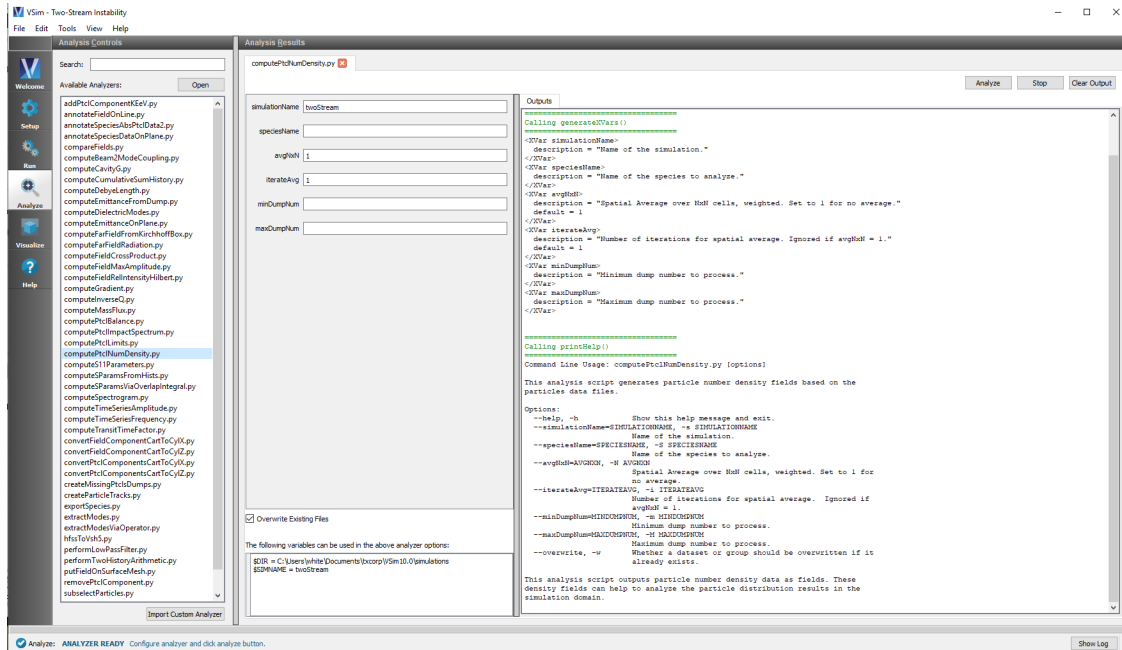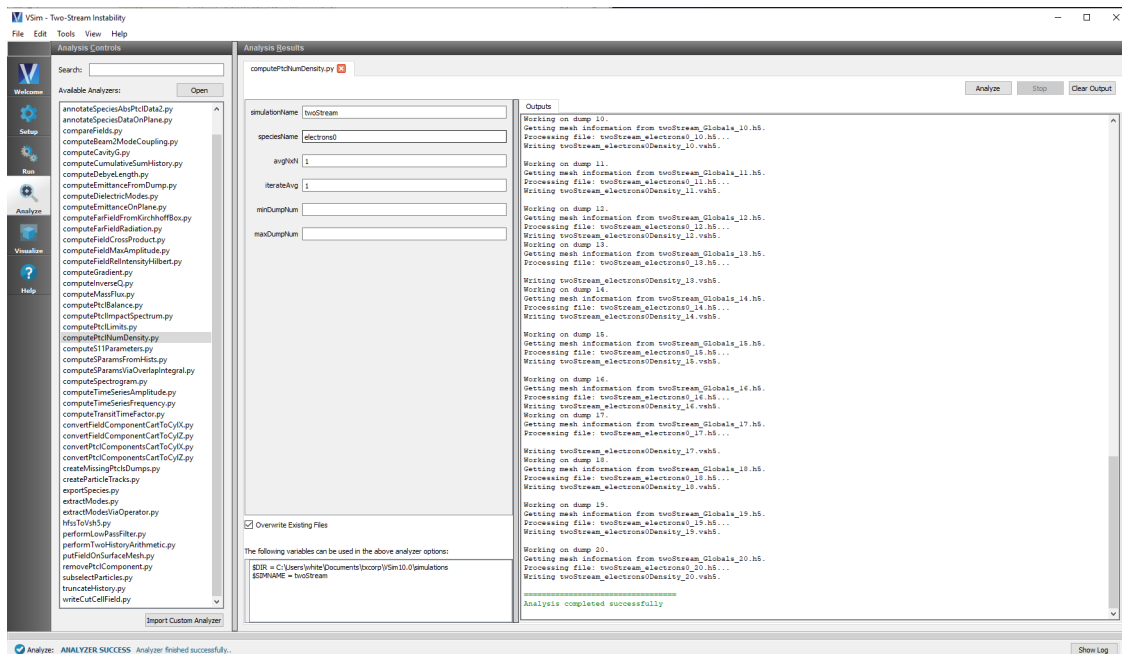
Fig. 10.2: Analysis Window with Parameters

Fig. 10.3: Analysis Window Successful Run

- For *Graph 1*, select *electrons0Density* for the Base Variable from the drop-down menu under *Graph 1*.

- For *Graph 2*, select *electrons0Density* again for the Base Variable. In the *Visualization Results* pane (2nd graph), click the *Fourier Amplitudes* box to see the Fast Fourier Transform output of the frequency domain.

- To match the visualization of this documentation, select *Dump 2* in the slide bar at the bottom of the *Visualization Results* pane.

You can explore other visualization options in this window, or can rerun the simulation with different parameters to investigate further.
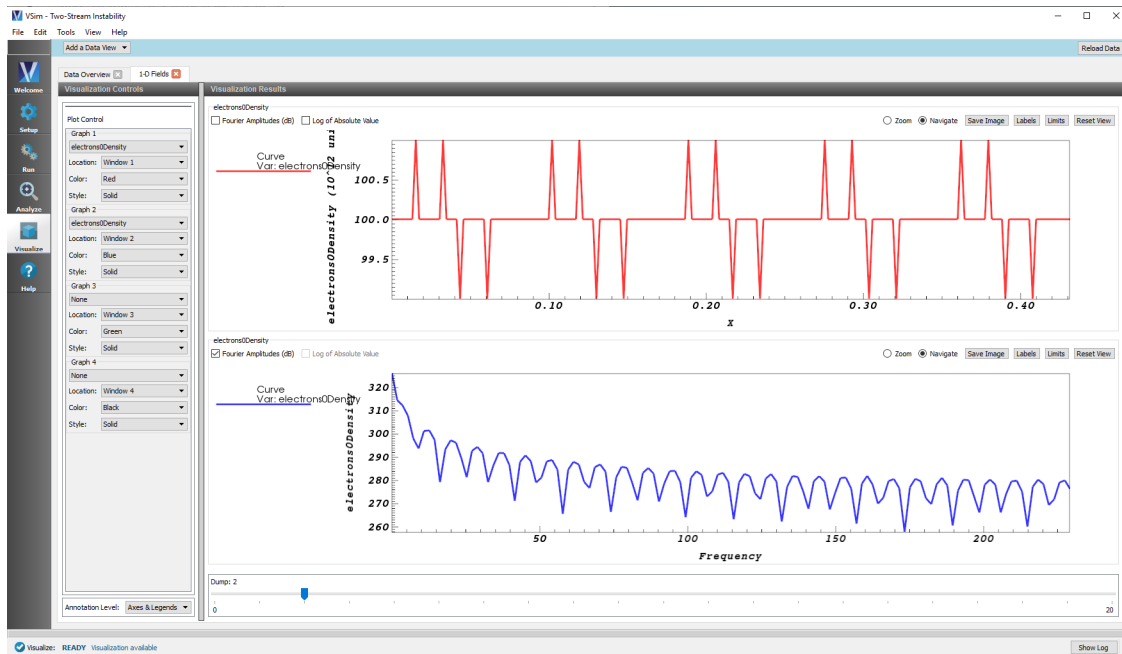

Fig. 10.4: Visualization of the analysis data

VISUALIZATION

11.1 Introduction to the Visualize Window

VSimComposer's Visualization feature is a flexible and comprehensive model viewer based on VisIt. The simulation tutorials and examples in *VSim Examples* provide several examples of using the Visualization feature's options in context.

The VSimComposer visualization tool is context sensitive, meaning that only those features that can be used with the current data are made available from the interface.

For more information on VisIt, please see: https://wci.llnl.gov/codes/visit/ and http://www.visitusers.org/index.php?title=VisIt_Wiki

For more information on using the VisIt context menu see: *Tools/VSimComposer Menu: Visualization Options*.

The Visualization window is divided into a *Visualization Controls* pane on the left and a *Visualization Results* pane on the right.

The type of data available to view is governed by the *Data View* pull down menu.

11.2 Select the Visualize Icon from the Icon Panel

Upon successful completion of the simulation run, the last message in the *Engine Log* tab is a reminder that you can now select the **Visualize** icon from the icon panel on the far left of the VSimComposer window as seen in Fig. 11.1.

11.3 Data View Pull-down Menu

In the top left of the main pane, you may select the kind of analysis that is to be performed. Again, this menu is context sensitive, so not all options may be available for your simulation. For example, you may only choose phase space if you have particles, and you may only paint fields onto surfaces if you have complex boundaries specified in `GridBoundary` blocks.

- Data Overview
- 1-D Fields
- Field Analysis
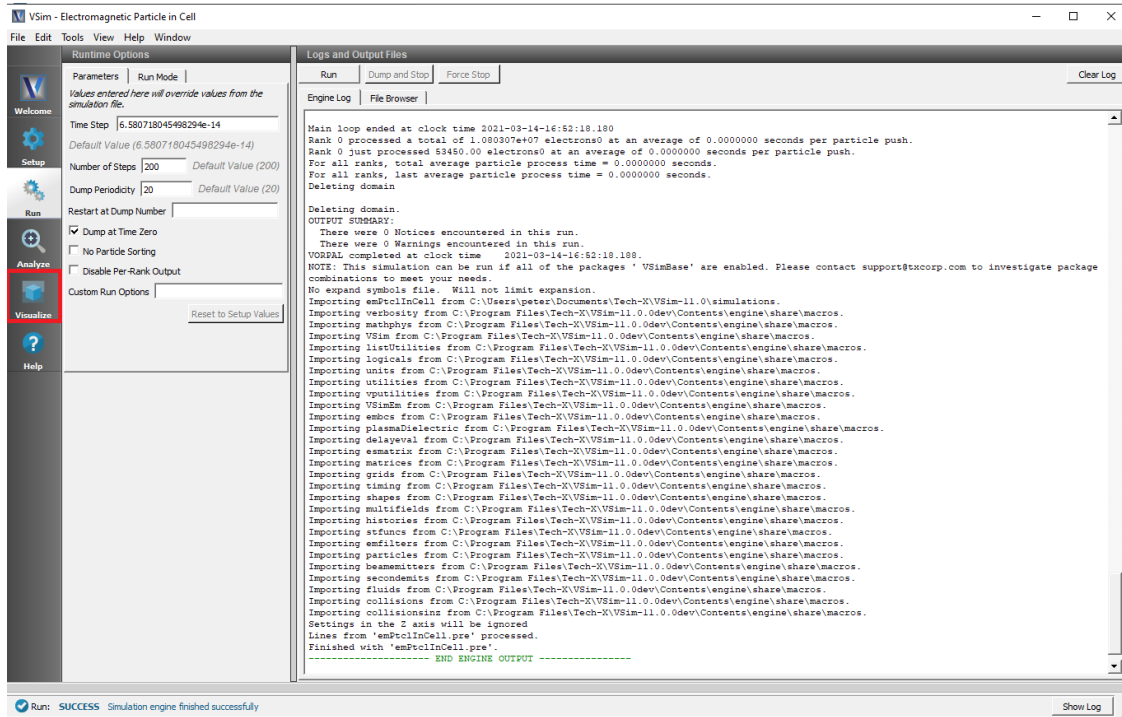- History
- Phase Space
- Binning

Fig. 11.1: Visualize Icon in Icon Panel

- Paint Fields

See Fig. 11.2.

11.4 Standard Controls Available Across Multiple Views

Several control buttons or choices are available across several different *Data Views*. They have the same functionality in each case and are documented below.

Annotation Level

To adjust the Annotation Level, use the *Annotation Level* drop-down menu at the lower left of the *Visualization Controls* pane.

- No annotations

- Axes only

- Axes & Legends

- All annotations

See Fig. 11.3.

Reload Data

You can visualize data from a simulation run as soon as it becomes available. If you decide to visualize data before a run is complete by switching to the **Visualize** tab, the VSim engine continues creating data files in the background.
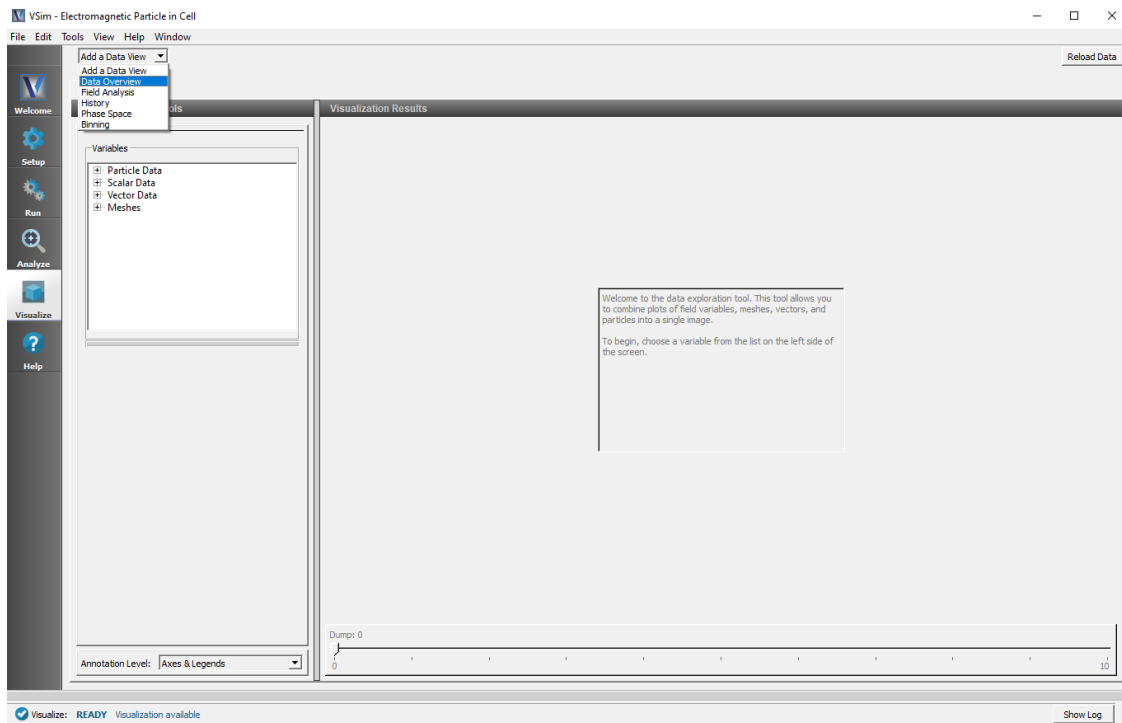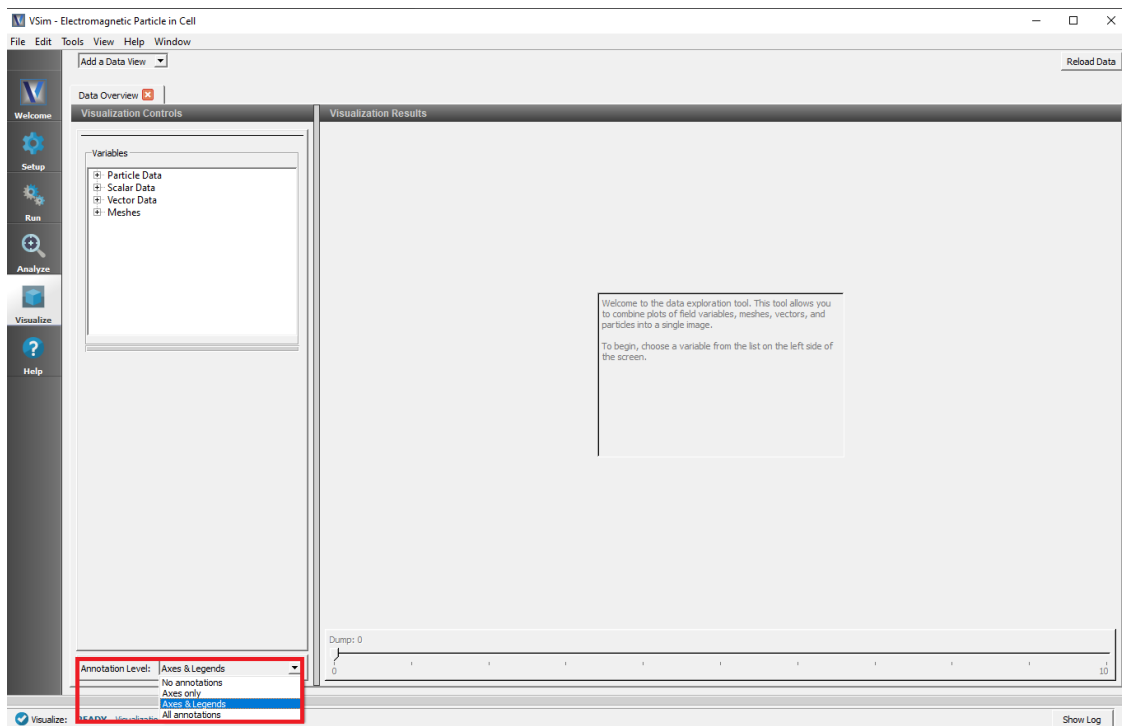
Fig. 11.2: Data View Menu

Fig. 11.3: Annotation Level

11.4. Standard Controls Available Across Multiple Views 153

Later, when more data is available for visualization or the simulation run is complete, use the *Reload Data* button in the top right-hand corner to visualize the new data. See Fig. 11.4.
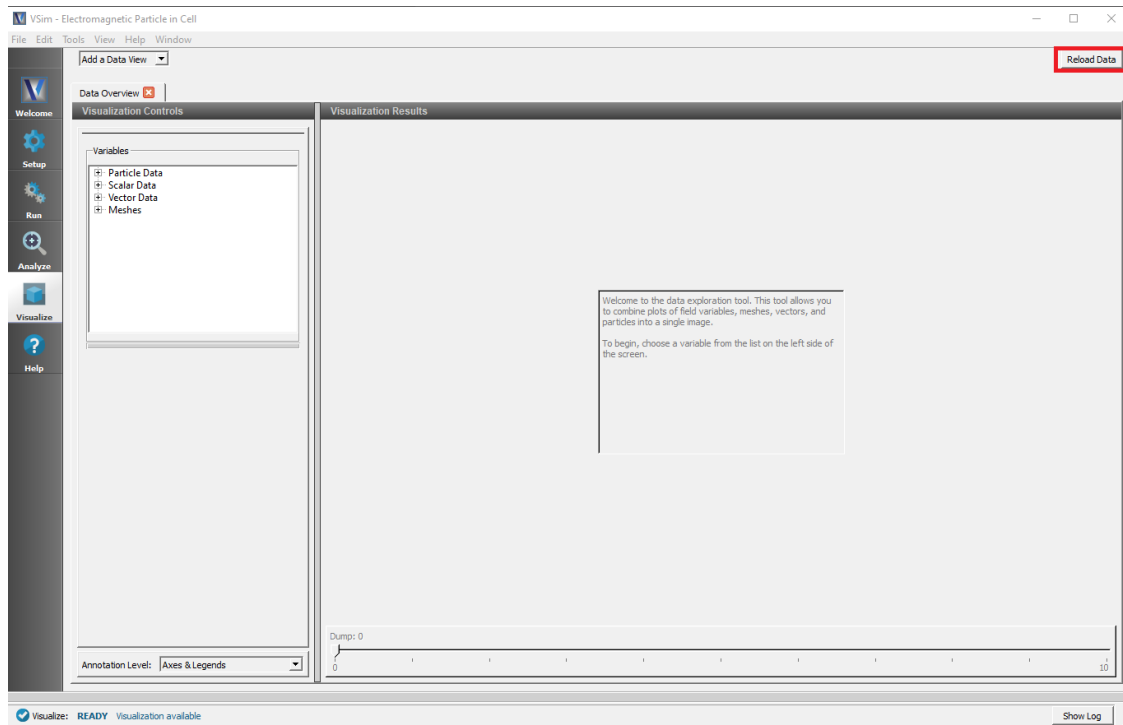

Fig. 11.4: Controls Pane Buttons

Save Image

This button saves the current image to your computer. You will be given options on where to save, the file name, and format as well as some options on size and dimension.

Labels

This brings up the *Axis Labels* window. See Fig. 11.5.
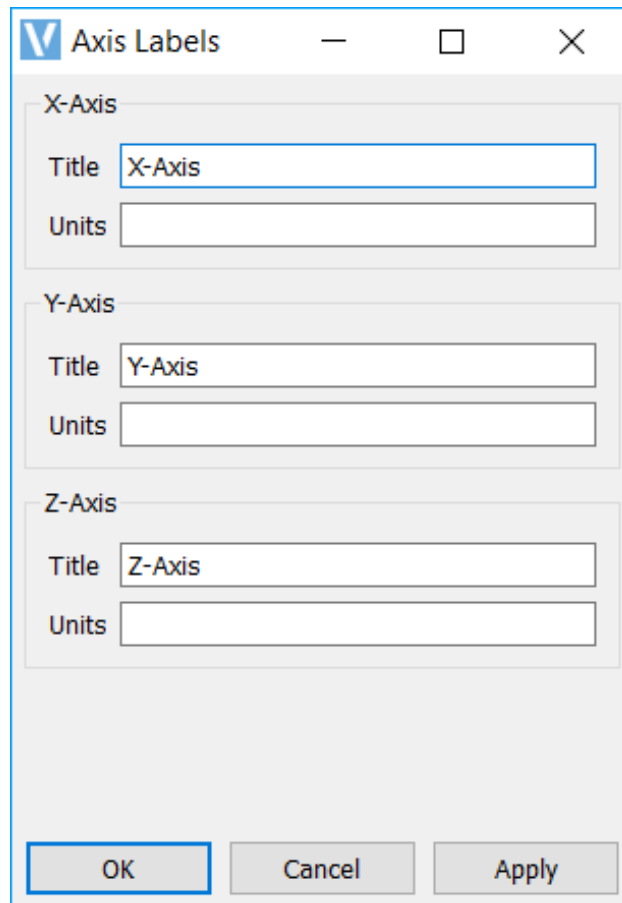
Limits

This button enables adjusting the *Limits* for each axis. See Fig. 11.6.

Rendering

This button allows for adjustment of the lighting and stereo effects. See Fig. 11.7.

Reset View

This button returns the objects in the *Visualization Results* pane back to their original location.
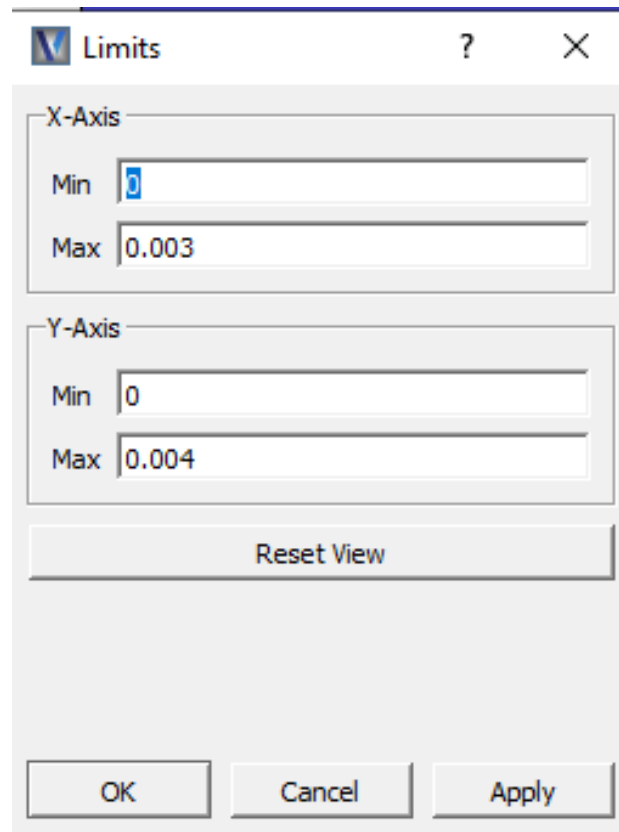
Fig. 11.5: Axis Labels Menu
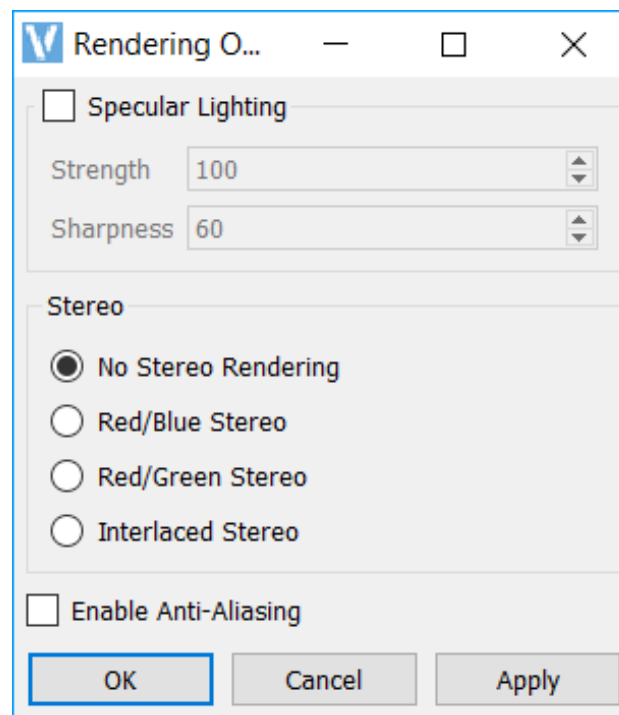
Fig. 11.6: Axis Limits Menu

Fig. 11.7: Rendering Menu

Auto Reset

Checking the *Auto Reset* box will force a *Reset View* each time the dump slider is moved.

Dump Slider

The slider at the bottom of the *Visualization Results* pane allows the user to move through the simulation results in time. Only the times for which files were "dumped" can be viewed.

11.5 Data Overview

Variables

The *Variables* section of the *Visualization Controls* pane enables you to choose which aspects of the simulation data to visualize. The types of variables that are available in the *Variables* section are dependent on your particular simulation. Below are some typically available types of variables for a 3D simulation containing fields and particles.

Particle Data

Types of *Particle Data* may include any *Species* in the input file:

- electrons

- ions

- neutrals

The base *species name* variable will show the particle position. The _x, _y, _z (or _r _z) variable will show the particle at it's position, with the value assigned being the coordinate in the respective direction.

The _ux, _uy, _uz, _ur, _uphi, variables will also plot the particle position, with the value assigned being the relativistic velocity in that axial direction.

If the simulation has been run in cylindrical coordinates, a {particleSpecies}_transform set of variables will be available. At this time the _transform variables will be equivalent, this transformation is used for non-uniform cylindrical grids, which are not currently available in visual setup.

Scalar Data

Types of *Scalar Data* may include fields like:

- E

- B

- J

Vector Data

Types of *Vector Data* include:

- E

- B

- J

Meshes

Types of *Meshes* include:

- globalGridGlobal (B)

- globalGridGlobal (E)
- globalGridGlobal (J)
- globalGridGlobal (coax)

See Fig. 11.8.
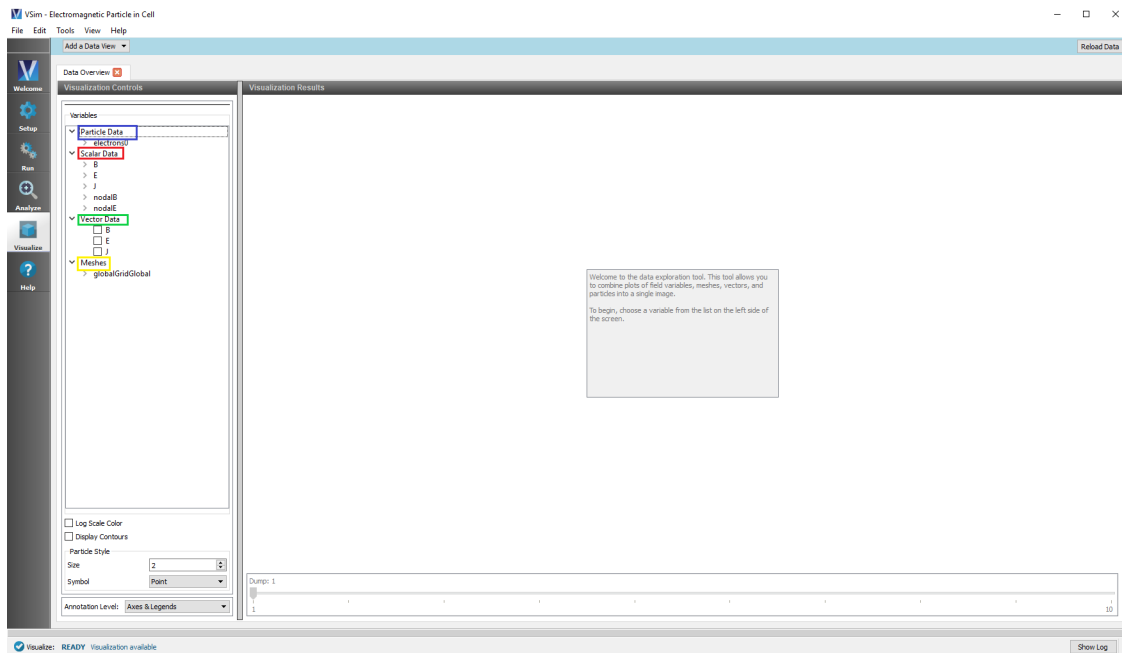
Fig. 11.8: Particle, Scalar, Mesh Data Variables

Geometries

Types of *Geometries* include:

- poly A wireframe of the part
- poly_surface A solid view of the part
- MappedPolysData A wireframe of the part
- MappedPolysData_surface A solid view of the part
- PecShapes All geometries in the simulation assigned PEC Material

All geometries can be clipped simultaneously by selecting the *Clip plot* checkbox, with the plane set by selecting *Plane Controls* It is also possible to change the opacity of a geometry usin the *Opacity* slider.

See Fig. 11.9.

11.5.1 Context Sensitive Options

The below options are available to each individual selection of scalar, particle or vector data. They may be applied to all pieces of scalar/particle data simultaneously by being set in the *Scalar Data* tab, or they can be applied on an individual basis, allowing for different settings for each piece of concurrently visualized data.
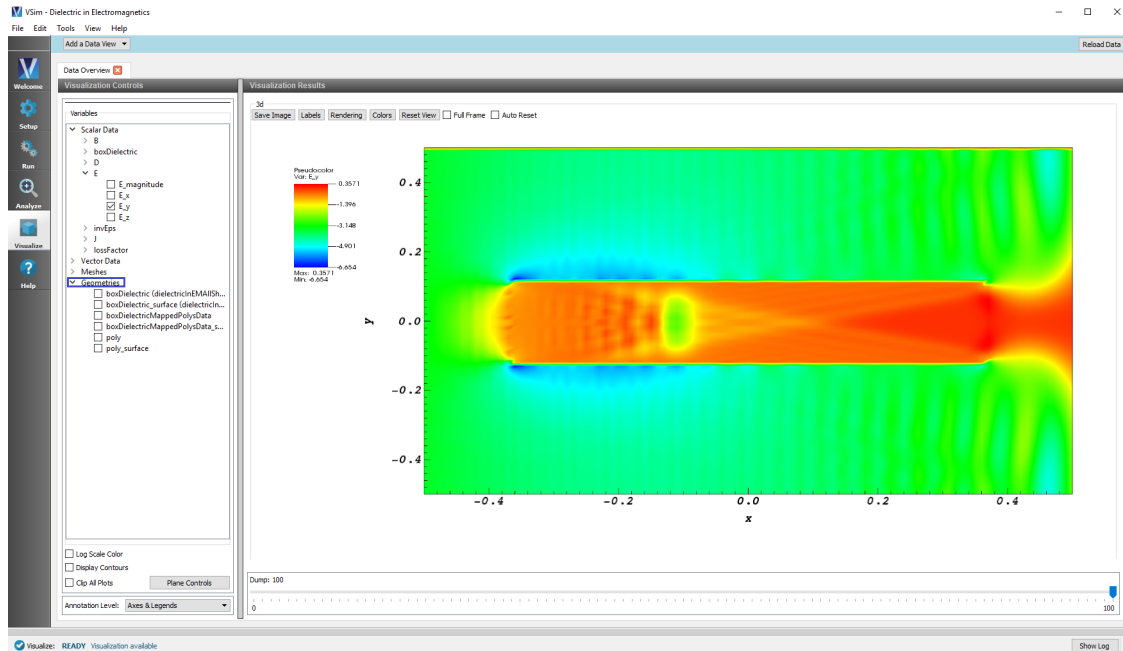
Fig. 11.9: Geometry Data Variables

Log Scale Color

If the appropriate field is selected, the *Log Scale Color* checkbox will be available to enable and disable display of log scale color.

Checking this box will put the color on a log scale. This is useful to see details in a field. See Fig. 11.10.

Display Contours

The default value in the *Contours* field is 5. If you select the Display Contours check box and have an appropriate data set selected, the number of contours can be changed to any positive number. See Fig. 11.11.

Clip Plot Checkbox

If the appropriate data is selected, the *Clip Plot* checkbox will be available to enable and disable plot clipping.

When *Clip Plot* is checked, the *Plane Controls* can be used to select an axis intercept or normal vector for the clipping. See Fig. 11.12.

Opacity

The opacity of any object in the visualization window can be adjusted, sliding the *Opacity* slider to the left will reduce the opacity, to the right will increase the opacity.

Color Table

The color table can be selected for each piece of visualized data, though *hot_desaturated* is the standard.
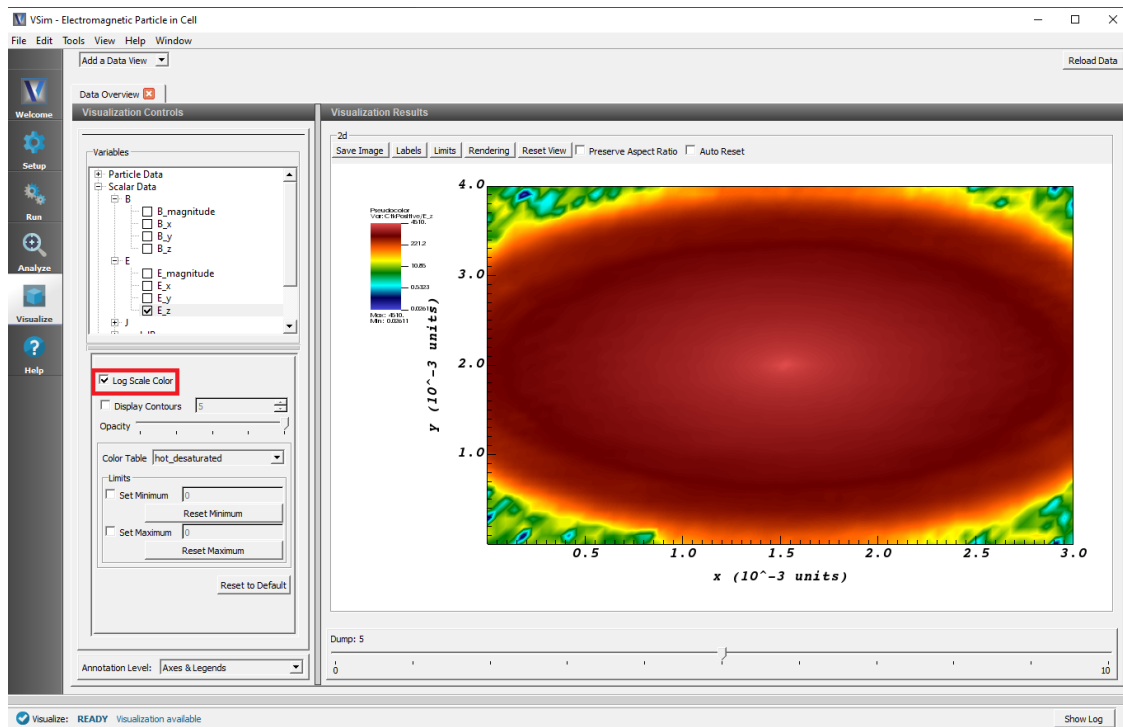
Fig. 11.10: Log Scale Color Checkbox
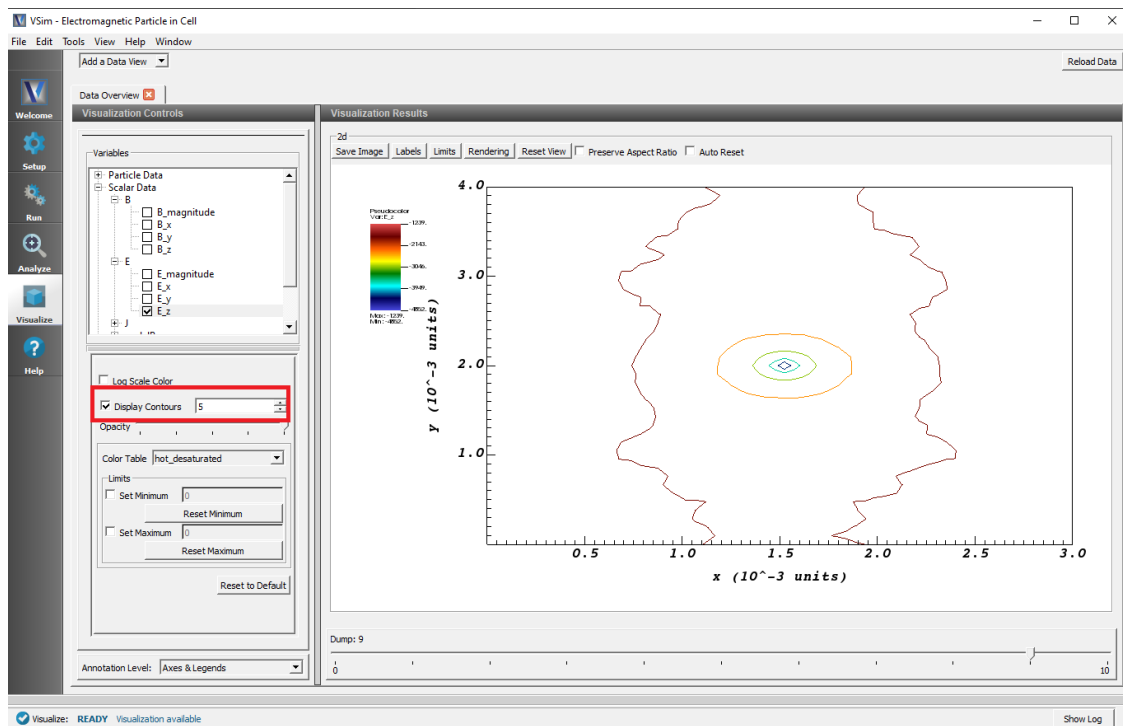
Fig. 11.11: Contours
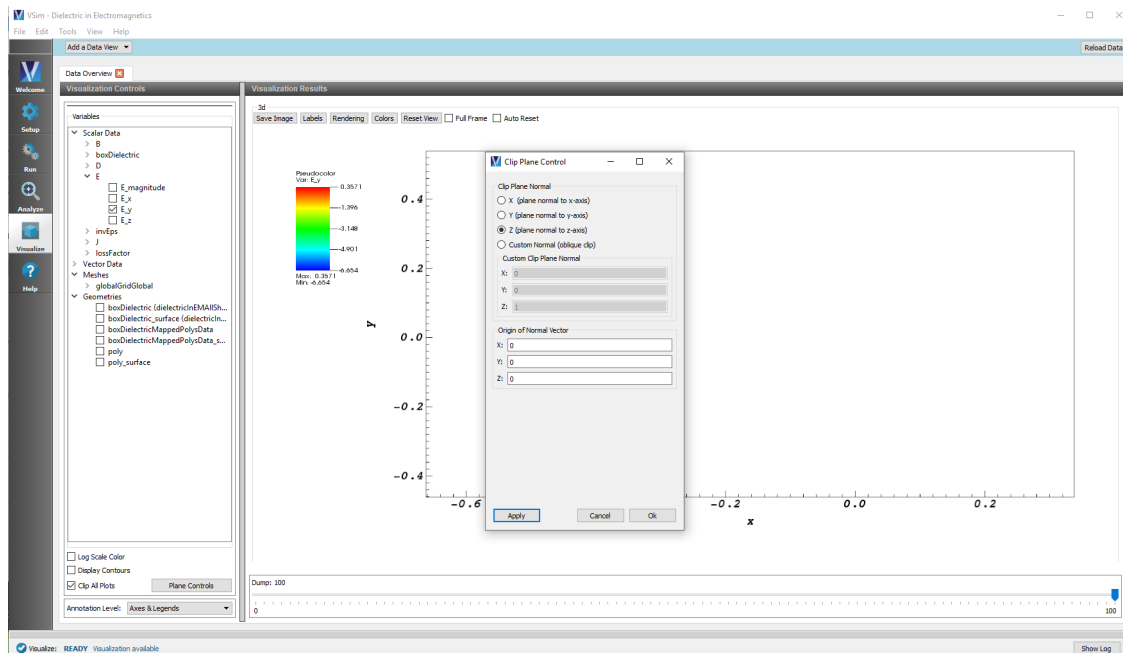
Fig. 11.12: Clip Plot Checkbox

Minimum and maximum values can be set by hand, by default they are scaled to the minimum and maximum values of that dataset.

Particle Style

The *Particle Style* section of the *Visualization Controls* panel enables you to control the appearance of particles in the visualization.

The *Size* field contains the size of each particle symbol.

The *Symbol* pulldown menu contains choices for the following shapes:

- Point (default)
- Box
- Axis
- Icosahedron
- Octahedron
- Tetrahedron
- Sphere Geometry
- Sphere

See Fig. 11.13.

It is also possible to set a specific flat color for the particles, selected with the *Set Color* button.
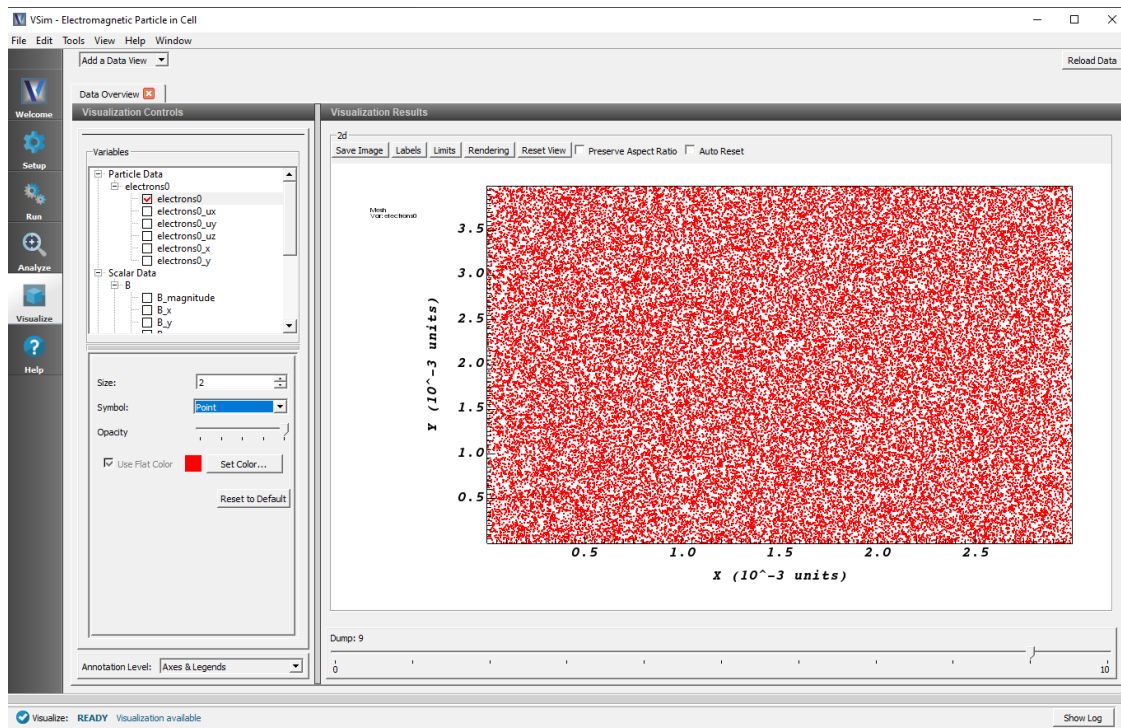
Fig. 11.13: Particle Style

11.6 Field Analysis

The *Field Analysis* data view will allow further analyzing of a particular field. From the *Add Data View* drop-down menu in the upper left corner of the pane, select *Field Analysis* . The *Visualization Controls* pane allows for selection of the field and selecting the line out location. The *Visualization Results* pane contains a 2d view of the chosen field, and a lineout at the selected location.

Field

The Field drop-down menu will allow you to choose which field from your simulation to do further analysis on. See Fig. 11.14.

Slice Settings

Slice settings will appear when a 3D field is viewed. The slice settings allow you to set the position of a slice of the 3D field to create a 2D plot. The *Plane Controls* botton allows for further control, including creating a slice at an angle. See Fig. 11.15.

Log Scale Color Table

Checking this box will put the color on a log scale. This is useful to see details in a field. See Fig. 11.16.
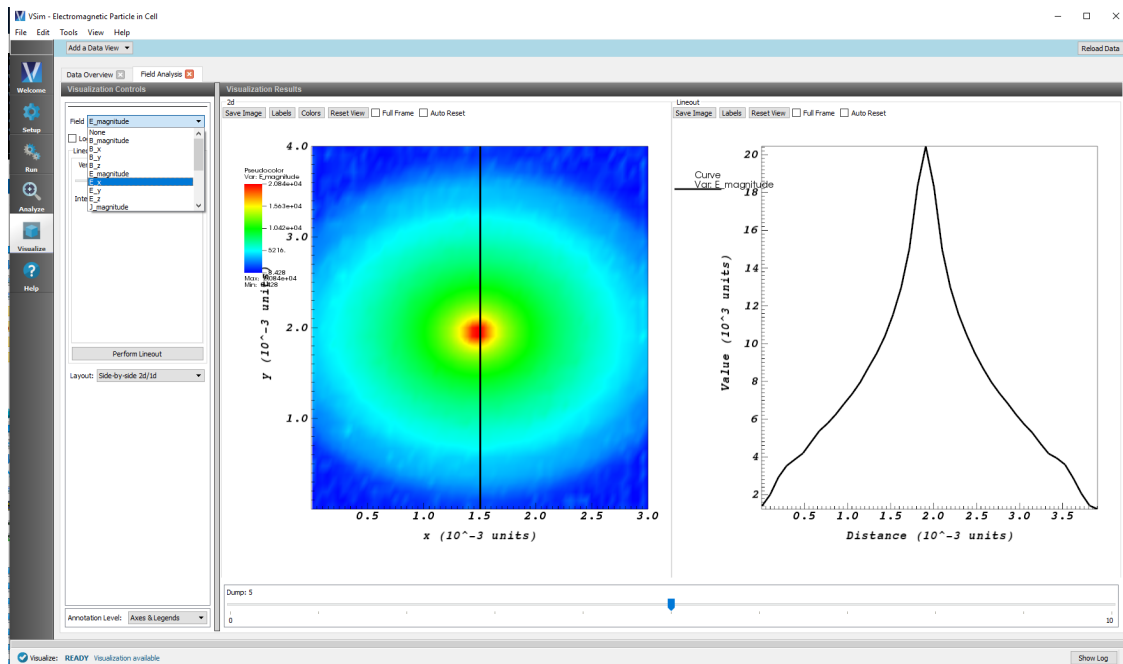
Fig. 11.14: Field drop down
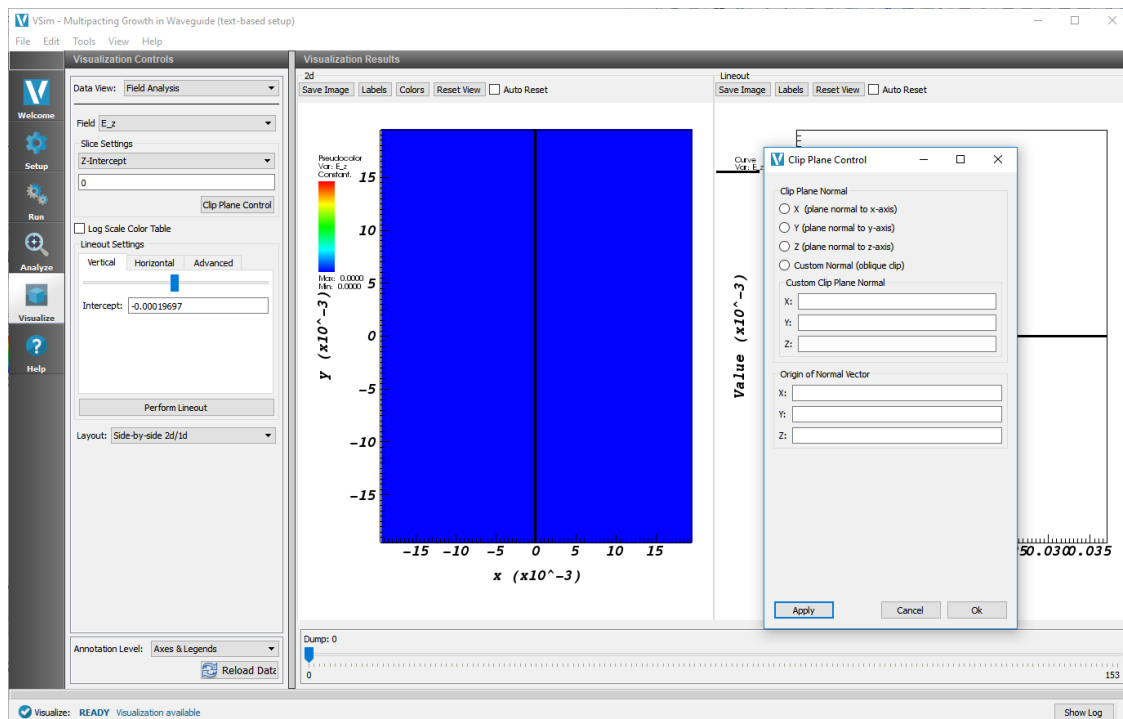
Fig. 11.15: Slice settings and plane controls for a 3D plot
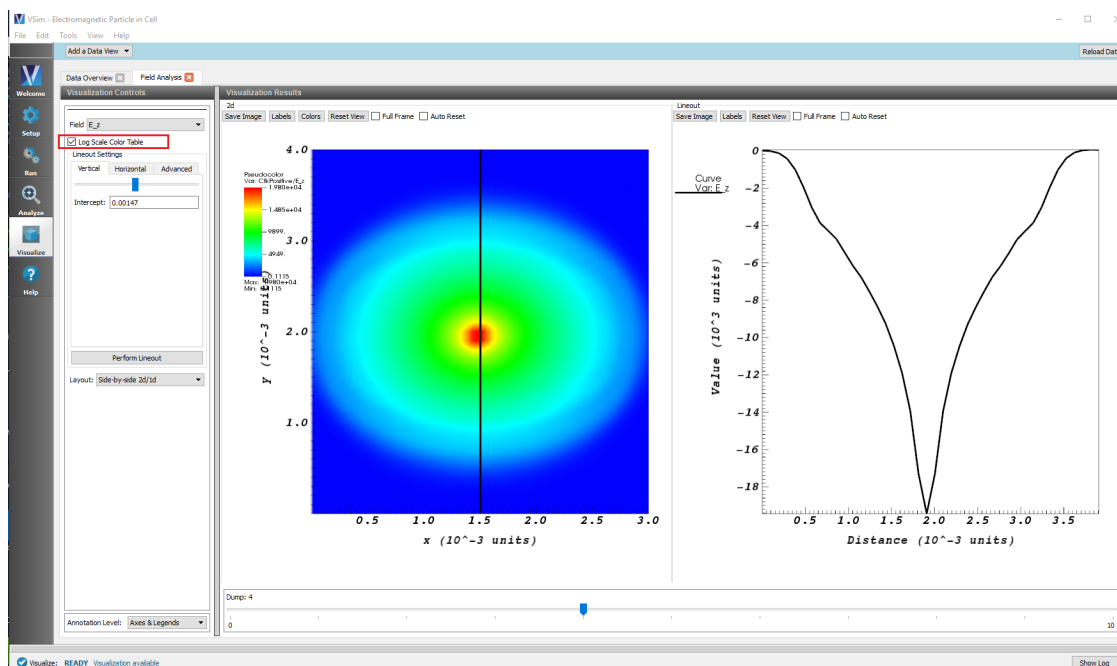
Fig. 11.16: Log Scale Color Table Checkbox

Lineout Settings

The position of the lineout can be changed using the *Lineout Settings*. The lineout can easily be set to vertical or horizontal at a specified intercept location, or the *Advanced* tab can be used to set a line in any arbitrary direction or length.

After setting the desired location of the lineout, press the *Perform Lineout* button to replot the lineout. See Fig. 11.17.

Layout

The layout of the *Visualization Results* pane can be changed from the default of Side-by-side 2d/1d. Options include:

- Side-by-side 2d/1d
- Stacked 2d/1d
- 2d Only
- 1d Only

See Fig. 11.18.

11.7 History

The *History* data view allows for plotting of any 1D array histories that were included in your input file prior to running your simulation. You can select this from the *Add Data View* drop-down menu at the top left-hand corner of the pane.

Up to 4 histories can be viewed at one time.
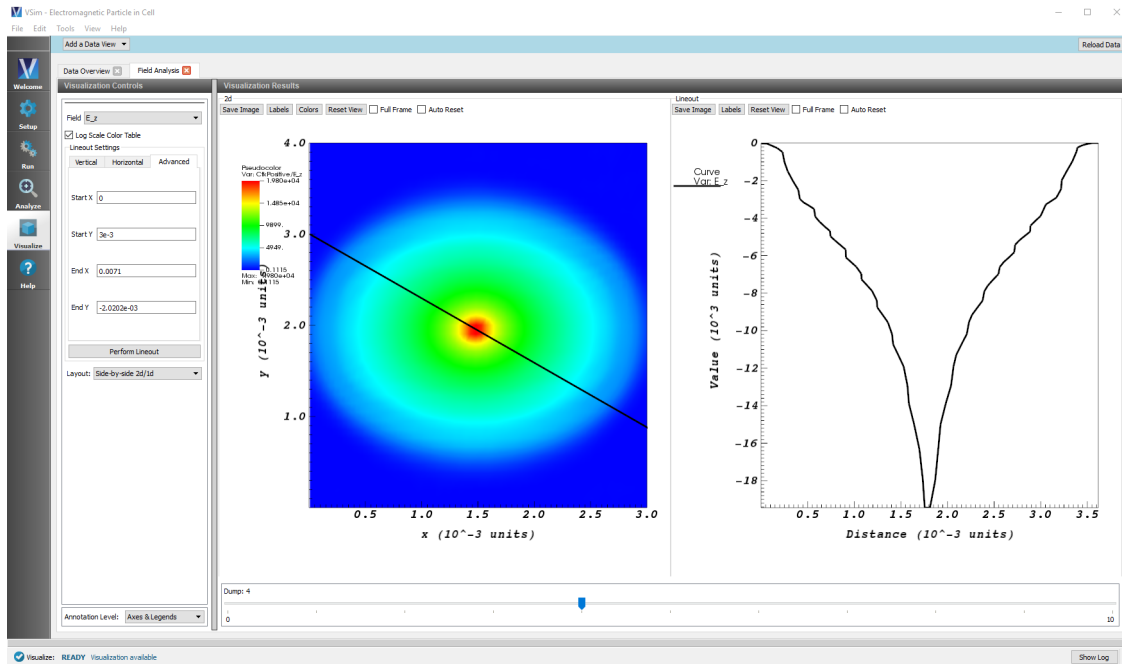
Fig. 11.17: The lineout settings controls
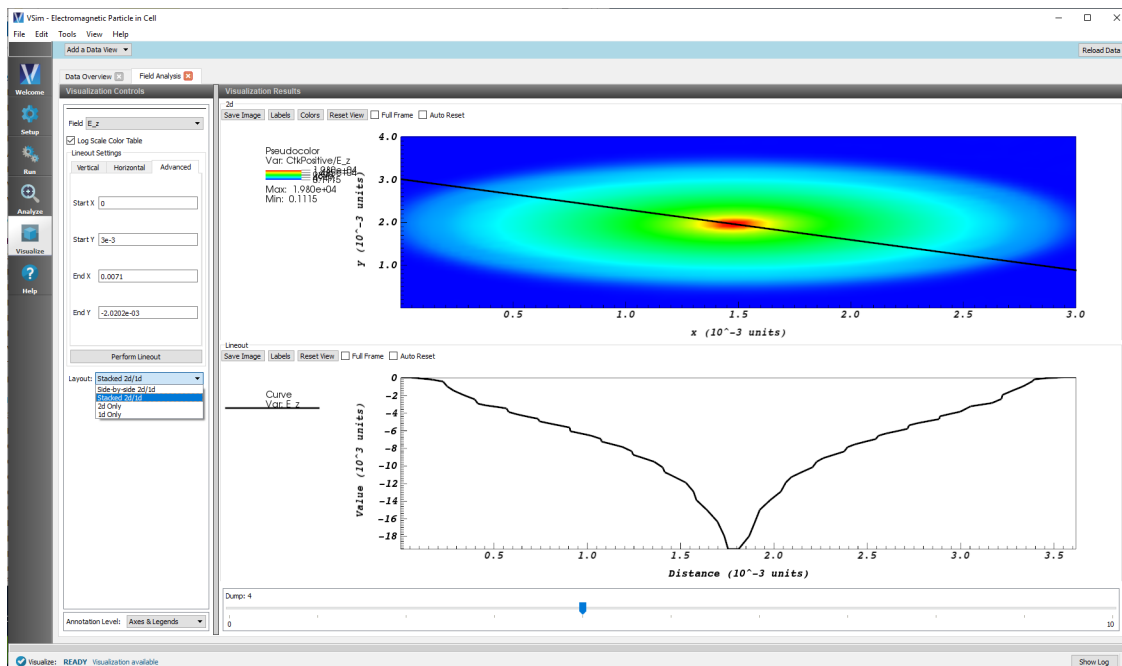
Fig. 11.18: The layout dropdown options

Location

The location drop-down allows you to plot multiple histories on top of each other. By setting the location of graph 1 and graph 2 to *Window 1*, both plots will appear in the same window.

Color

The color of the line can be modified using the *Color* drop down. Choices include red, blue, green, and black.

Style

The style of the line can be modified using the *Style* drop down. Choices include solid, dash, dot, dotdash, and points.

Fourier Amplitudes (dB)

Selecting the *Fourier Amplitudes* checkbox will take a Fast Fourier Transform of the data.

Log of Absolute Value

Selecting the *Log of Absolute Value* checkbox will put the y-axis on a log scale.

Zoom

Setting the *Zoom* radial selection will switch the mouse click feature to zoom. Start by clicking the mouse inside the plot window and then dragging to create a rectangle. Finish by un-clicking the mouse button. The plot will be zoomed to the data contained inside the rectangle.

Navigate

Setting the *Navigate* radial selection will switch the mouse click feature to navigate. Click the mouse inside the plot window and drag to move the plot. See Fig. 11.19.

11.8 Phase Space

The *Phase Space* data view allows for plotting of any particles (species) in your simulation. You can select this from the *Add Data View* drop-down menu at the top left-hand corner of the pane.

Base Variable

The *Base Variable* can be used to switch between any of the particle species in your simulation.

X-axis

The variable to be plotted on the x-axis.
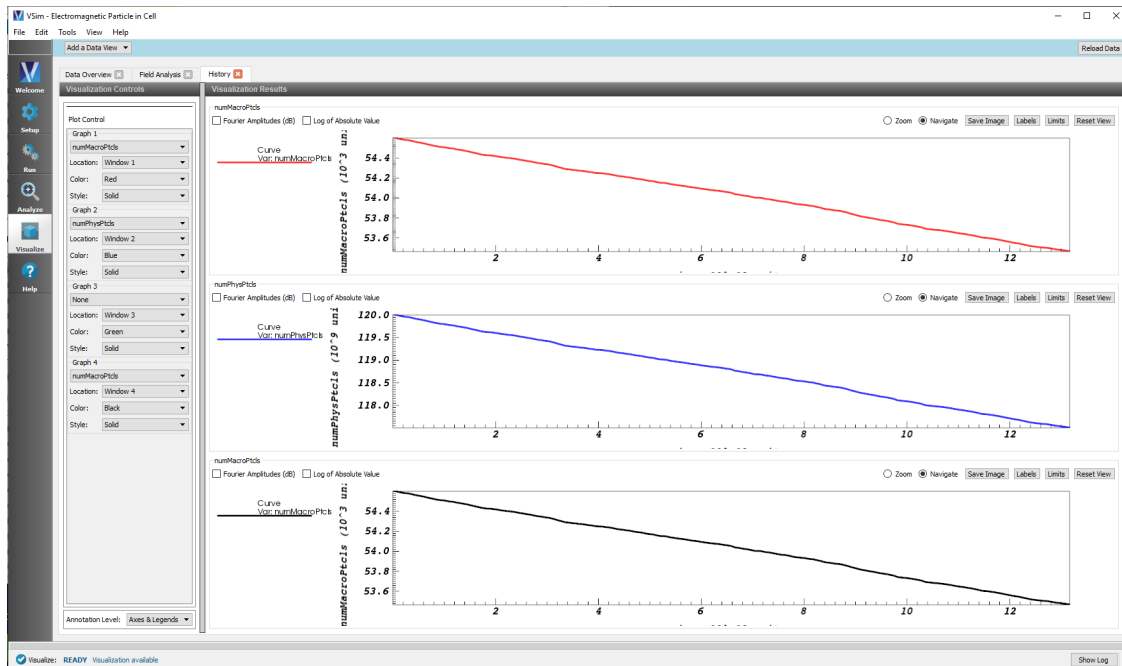
Fig. 11.19: The History View

Y-axis

The variable to be plotted on the y-axis.

Z-axis

The variable to be plotted on the z-axis.

Color

The particles can be plotted in either a solid color such as red, green, or blue, or they can be plotted using another variable as their color. An example is to plot the velocity as the color on an x, y, z spatial plot.

Point Size (pixels)

The size of each particle symbol.

Symbol

The *Symbol* pulldown menu contains choices for the following shapes:

- Point (default)
- Box
- Axis
- Icosahedron

- Octahedron

- Tetrahedron

- Sphere Geometry

- Sphere

Enable Second (Third) Plot

Up to 3 particle species can be plotted at one time in the *Phase Space* window. To enable another plot, check the box.

Reset View on Draw

When changes are made to the variables to each plot, you must click the *Draw* button. Check the *Reset View on Draw* button if you would like the view reset each time the draw button is clicked.

Draw

When changes are made to the variables to each plot, you must click the *Draw* button to redraw the plot.

See Fig. 11.20.
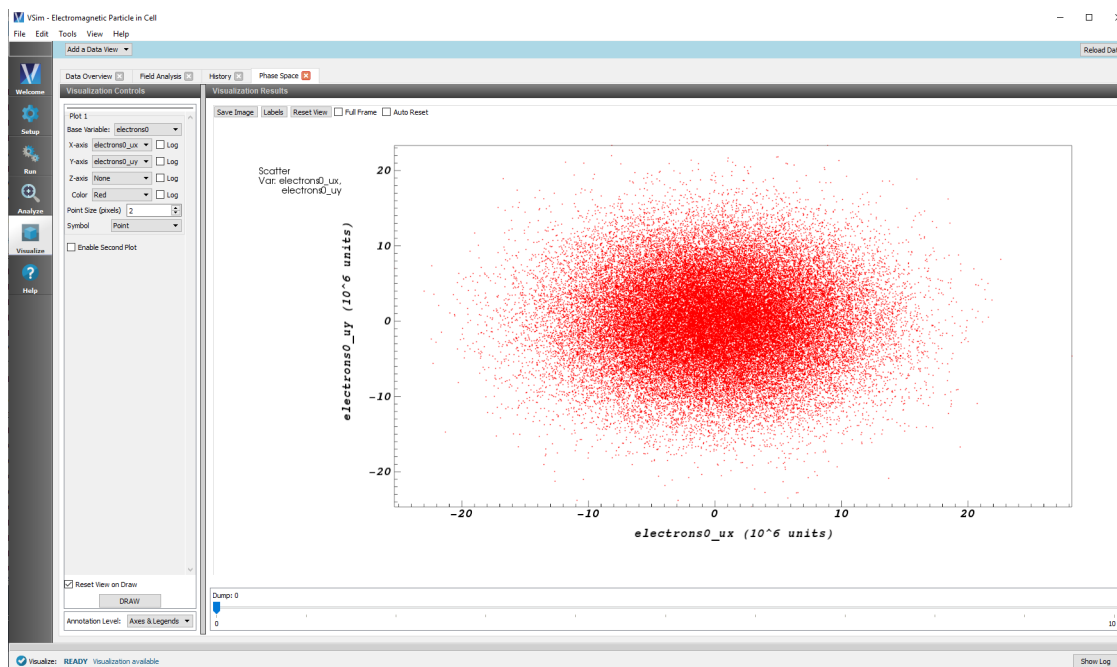
Fig. 11.20: The Phase Space View

11.9 Binning

The *Binning* data view allows for "binning" the particles (species) in your simulation and creating histogram-style plots. You can select this from the *Add Data View* drop-down menu at the top left-hand corner of the pane.

Base Variable

The *Base Variable* can be used to switch between any of the particle species in your simulation.

Variable

The *Variable* to be binned.

Bins

The number of *Bins*. The variable will be divided into the number of bins chosen.

Reduction Operator

The *Operator* is the method used for binning.

Reduction Variable

If the *Variable* is active (depending on the operator), the variable what the operator acts on.

Draw

When changes are made to the variables, you must click the *Draw* button to redraw the plot.

Slicing

Slice settings will appear when a third dimension is added. The slice settings allow you to set the position of the slice of the 3D field to create the 2D plot. The *Plane Controls* button allows for further control, including creating a slice at an angle. See Fig. 11.21.
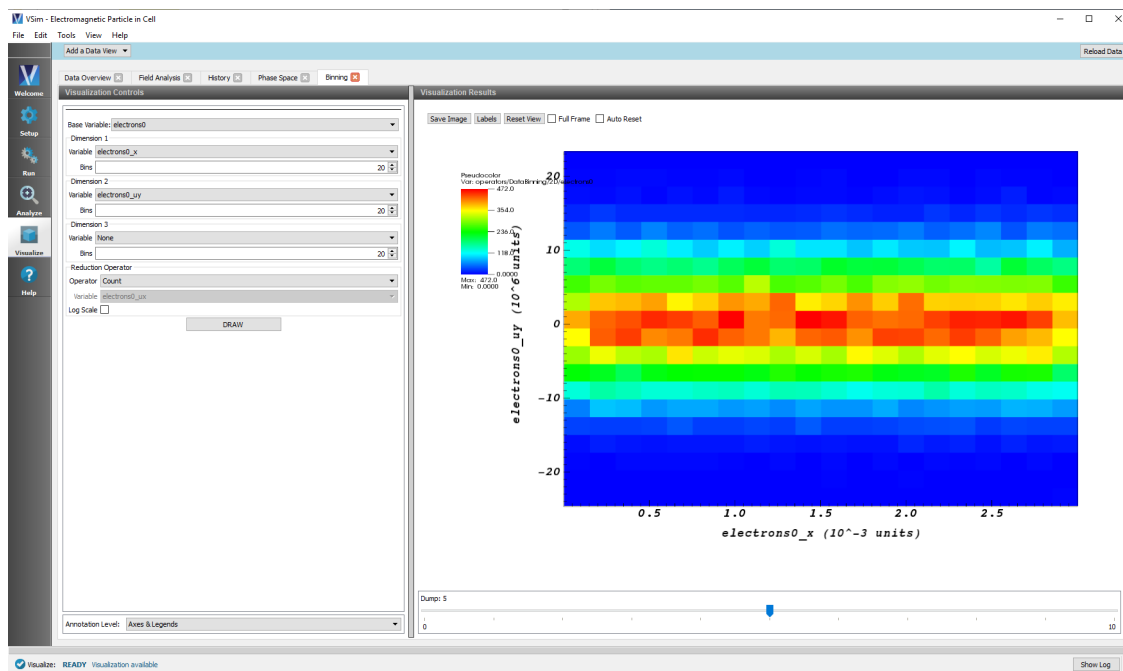
Fig. 11.21: The Binning View

TROUBLESHOOTING

12.1 Troubleshooting Electrostatic Simulations

12.1.1 Simulation Crashes at Startup with PEC Dirichlet Boundaries

PEC objects in the simulation must be entirely inside of the simulation grid in order to ensure proper problem setup.

12.1.2 The Simulation Does Not Finish Properly

The most common cause of crashes is improperly set up particle boundaries. The particle boundaries must completely surround the space in which particles are loaded. Otherwise particles can drift out of the grid and try to reference fields that do not exist. This leads to a Vorpal segmentation fault.

Another possible reason for an electrostatic simulation not finishing properly is that a particle has crossed more than one cell in a time step. This could allow the particle to pass through a particle sink without being absorbed.

- One solution is to reduce the duration of the time step.

- Another solution is to limit the number of cells a particle can cross in one time step by artificially reducing the velocity of high speed particles. See VSimReferenceManual: Text Setup: Species: maxcellxing

It could be that the definition of the Particle Species is incorrect.

The following Species input block is not defined correctly:

```
<Species electrons>
  kind = nonRelBoris
  emField = myZeroField
  ...
</Species>
```

- The problem: The input block does not specify mass and charge.

- The result: The simulation runs normally with no complaints. The default mass and charge are those of a positron.

- The solution: Include the mass and charge of your species every time they are defined.

12.1.3 The Electrostatic Solver Does Not Converge

If the electrostatic solver does not converge, this often indicates a problem with the setup. The matrix can be singular in a fully periodic system due to the failure to specify the value of the potential at one point.

12.2 Troubleshooting Electromagnetic Simulations

12.2.1 The Simulation Does Not Finish Properly

The most common cause of crashes is improperly set up particle boundaries. The particle boundaries must completely surround the space in which particles are loaded. Otherwise particles can drift out of the grid and try to reference fields that do not exist. This leads to a Vorpal segmentation fault.

12.2.2 The Output Shows an Unexpected High-Frequency or Checkerboard Pattern

A common problem with electromagnetic simulation is not following the Courant condition [CFL28]. The Courant condition states roughly that the time step must be small enough that a light wave cannot cross more than one cell in a single time step.

High-Frequency and checkerboard patterns can be symptoms of an instability resulting from violating the Courant condition.

12.3 Troubleshooting Visual Setup Crashes

The Visual Setup makes setting up simulations much simpler than writing an input file, but the output messages when a simulation crashes can be mysterious. If a simulation crashes before the engine takes the first step, it is more than likely that there is an error in the `.sdf` file. Documented below are some common errors, and the output messages they will create at runtime. If your simulation is crashing before the first step, take a look to see if any of the error messages shown below matches yours.

If your error message is NOT below, send an email to `support@txcorp.com` and an Application Engineer will take a look at your simulation, and maybe your error will show up in the documentation for the next release!

12.3.1 General Tips

- Check the `.in` file for any unevaluated expressions.

- The message "Object %%%%%: Could not locate object * in the input file object hierarchy" usually means that something is missing from the `.sdf` file. Either a variable used in the simulation isn't present or is miss-named, or an attribute isn't set/is left blank. Try to figure out what the "%%%%%" object is in the simulation then figure out what is missing. See below for some specific examples the examples.

- Running in parallel can sometimes suppress error messages. If you get a crash try running in serial for a better error message.

- If a simulation with particles makes it to the first step, then hangs, it is possible that more particles than VSim can handle are being loaded into the simulation.

- Objects need to be assigned a material before they can be used else where in the simulation.

- Geometry objects cannot set boundary conditions on the walls of a simulation. That is, a boundary condition with the "boundary surface" attribute set to "shape surface" will fail to set a boundary on the upper x, lower x, upper y, etc surfaces of the simulation.

- When using a geometry object to set a boundary condition internal to the simulation boundaries, make the geometry extend beyond the nodes on which you wish to set the boundary.

12.3.2 Parameter/Constant Not Named Properly

The following error was created in a simulation that had a constant named *BEAM_RADIUS* but had a SpaceTimeFunction that used "BEAMRADIUS".

Here, the unknown expression ("undefined symbol") was printed out, providing an avenue for beginning the debugging process.

```
Sources are:
 BeamEmitter
 sputterNeutralEmitter0

ERROR: In setting up simulation:
TxsSymbolTable::checkUndefVar: Undefined symbol 'BEAMRADIUS'.

Settings in the Z axis will be ignored
Lines from 'ionBeamSputtering.pre' processed.
Finished with 'ionBeamSputtering.pre'.
Error building data.

-------------------- END ENGINE OUTPUT ----------------

Engine completed with error: VORPAL INPUT-FILE ERROR (code 5)
```

12.3.3 History Attribute Not Set

This error was from the same simulation as the error above. A history to count the number of physical particles was added, but the species was left blank.

Unlike the error above, there is no "ERROR: ..." statement printed out. The nature of the error has been highlighted in the image below for visibility. Sometimes, the nature of the error can be buried in the output, so keep a sharp eye out for statements like the one below, because it does (indirectly) point to the cause of the crash.

```
Source named BeamEmitter added to Species ArgonIons.
Source named sputterNeutralEmitter added to Species ArgonIons.
Building species neutralCopper with NDIM = 2.
Added sink, lower0Absorber, to global region:
[-1, 0) x [-1, 33)Added sink, upper0Absorber, to global region:
[32, 33) x [-1, 33)Added sink, lower1Absorber, to global region:
[0, 32) x [-1, 0)Added sink, upper1Absorber, to global region:
[0, 32) x [32, 33)ERROR: In setting up simulation:
Object numPhysPtcls0: Could not locate object please in the input file object hierarchy..

Settings in the Z axis will be ignored
Lines from 'ionBeamSputtering.pre' processed.
Finished with 'ionBeamSputtering.pre'.
Error building solvers.

-------------------- END ENGINE OUTPUT ----------------
```

12.3.4 Missing Attribute in Boundary Condition

This error is very similar to the one above (History Attribute Not Set). In this case a Dirichlet boundary condition named "dirichlet1" was added to the simulation, but a surface for the boundary condition was not specified.

The name of the boundary condition that was added was "dirichlet1". In the image below, notice that this name appears in the error message ("Object dirichlet1Filler ...").

12.3.5 Over-Specifying Boundary Conditions

This error arose when two, overlapping CSG geometry objects were used to set the voltage on a region in the simulation, so the voltage was over-specified.

```
ERROR: In setting up simulation:
Problem found setting up MultiField:
Object dirichlet1Filler: Could not locate object selectSurface in the input file object hierarchy..

Settings in the Z axis will be ignored
Lines from 'ionBeamSputtering.pre' processed.
Finished with 'ionBeamSputtering.pre'.
Error building solvers.

-------------------- END ENGINE OUTPUT ----------------
```

Depending on which electrostatic solver is being used, a different message will be output. The first error message below was the result when using the "superLU" direct solver:

```
Lines from 'ionBeamSputtering.pre' processed.
Finished with 'ionBeamSputtering.pre'.
ERROR: An unhandled exception was raised: Error!  The parameter "Total numeric factorization time" does not exist
in the parameter (sub)list "ANONYMOUS".

The current parameters set in (sub)list "ANONYMOUS" are:

  {
    "Total matrix redistribution time" : double = 0
    "Total Amesos overhead time" : double = 0
    "Total symbolic factorization time" : double = 0
  }


Throw number = 1



-------------------- END ENGINE OUTPUT ----------------
```

This message was output when using the "gmres" iterative solver:

```
Species neutralCopper has velocity limits = [929072, 464536].
Sinks are: lower0Absorber upper0Absorber lower1Absorber upper1Absorber
---------------------------------------------------------------------------
Using `power-method' for eigen-computations
***
*** ML_Epetra::MultiLevelPreconditioner
***
Matrix has 1103 rows and 4699 nonzeros, distributed over 1 process(es)
The linear system matrix is an Epetra_CrsMatrix
** Transforming column map of Main linear system matrix
Default values for `DD'
Maximum number of levels = 10
Using increasing levels. Finest level  = 0, coarsest level = 9
Number of applications of the ML cycle = 1
Number of PDE equations = 1
Aggregation threshold = 0
Max coarse size = 128
R and P smoothing : P = (I-\omega A) P_t, R = P^T
R and P smoothing : \omega = 1.333/lambda_max
Null space type       = default (constants)
Null space dimension = 1
-------------------- END ENGINE OUTPUT ----------------
```

12.3.6 Under-Specifying Boundary Conditions

This is an example of an error where the printed error statement provided little to no help in debugging, unless you are deeply familiar with matrix solver packages (Epetra constructs matrices and is the part of the Trilinos Library which is used by VSim).

The cause of this error is that there are insufficient boundary conditions resulting in an in complete matrix. If you see this error, adjust your boundary conditions to make sure boundary conditions are completely set on all boundaries of the simulation Note: geometries cannot be used to set boundary conditions on the walls/boundaries (lower x, upper x, lower y, etc.) of a simulation.

```
ERROR: In setting up simulation:
Problem found setting up MultiField:
Epetra error -1 occurred calling FillComplete() on matrix..

Settings in the Z axis will be ignored
Lines from 'ionBeamSputtering.pre' processed.
Finished with 'ionBeamSputtering.pre'.
Error building solvers.

-------------------- END ENGINE OUTPUT ----------------
```

12.3.7 Particle Loader Missing Loading Volume

Below is an example of what happens when a particle loader is not setup correctly. This is another more mysterious error message, since after the "ERROR: In Setting up simulation" there is very little to indicate the location of the bug. However, notice that the last thing printed before the error was "Sources are: /n particleLoader0". This is a hint as to where to check your input file.

```
Sources are:
 particleLoader0

ERROR: In setting up simulation:
No prmVec named 'lowerBounds'.

Settings in the Z axis will be ignored
Lines from 'ionBeamSputtering.pre' processed.
Finished with 'ionBeamSputtering.pre'.
Error building data.

-------------------- END ENGINE OUTPUT ----------------
```

12.3.8 Fluid Missing Volume

This error is almost identical to the particle loader error listed above. A background fluid was added to the simulation, and the volume attribute was not set. Notice how the simulation crashes right after printing out a statement about the "neutralFluid1".

```
[0, 32) x [-1, 0)Added sink, upperlAbsorber, to global region:
[0, 32) x [32, 33)neutralFluid1 has the grid initial conditions: initialDensity.
ERROR: In setting up simulation:
No prmVec named 'lowerBounds'.

Settings in the Z axis will be ignored
Lines from 'ionBeamSputtering.pre' processed.
Finished with 'ionBeamSputtering.pre'.
Error building solvers.

-------------------- END ENGINE OUTPUT ----------------
```

12.3.9 Unable to Open SDF (No Such Attribute Error)

If after creating a simulation with an imported CAD file and changing the file import, it cannot reopen with an error "getAttrib const: No such attribute. Attributes are..." This can be caused by a CAD file with multiple parts having the same name, and then the number of parts with multiple names changing. For example if a single .step file is imported twice, and then a single deleted. This is due to an issue with how VSim handles parts with multiple identical names. It is not an issue if the number of imported CAD file's stays consistent.

To correct this issue contact Tech-X Support.

12.3.10 Geometries Having Poor Lighting

If your simulation uses geometries or grids smaller than 1mm or larger than 100m, click the 'Show Scale' button in the 3D view and use the scale slider to select the appropriate scale for rendering. This will prevent issues with geometry

visibility, axis labeling, and item coloring.

12.4 Troubleshooting Plasma Density

If *applyTimes* is set, and includes time t=0 in the range, particles may be loaded both during initialization and in the zeroth time step of the model, potentially giving twice the density you want.

Use together with a history of *kind=speciesNumPhysical* and compare the total population to the number of particles you expected.

Use together with the computePtclNumDensity script to check the overall density of the loaded particles.

12.5 Troubleshooting Missing Secondary Particles

If *suppressEnergy* is left to its default value of 0 in text driven simulations, this will prevent loading of secondary electrons subject to an E field which would accelerate it into the boundary. The user can try switching off the sources of E fields to check if the emission is correct. Due to space-charge it's quite likely that many of the secondary electrons will then experience the field and be driven into the boundary, having their emission suppressed. In other words they will not appear in the simulation. Setting *suppressEnergy* to a large value like $1.e20$ should address the issue.

See also the descriptions of Secondary Electron Emitter and secElec in the VSim Reference Manual.

12.6 Troubleshooting Crashes During Stepping of Particle Simulations

Once the simulation starts timestepping, most potential causes of premature termination of the simulation have been passed. If the simulation does end prematurely, there are several steps we can take to diagnose what is happening and speed up resolving whatever issues are causing these.

Firstly, there are several potential causes for crashes:

- The simulation is taking up too much memory.
- The simulation is trying to access areas of memory it should not.
- The simulation is looking for an object which it cannot find.
- The hard drive has run out of space.

If the simulation is taking up too much memory, it is a good idea to determine if the number of particles in the simulation is rising steadily or running away. Instrumenting a history to measure the number of macroparticles in each species is a good way to determine if this is the cause. Rerun the simulation to a few steps short of where the crash happened previously, then reduce the `dumpPeriodicity` to get more information about the steps where the crash is occuring. If the crash is due to a runaway of the number of macros inside the simulation volume, then it will often be preceded by a slowing down of the wall-clock time per step.

If the number of particles is reasonable, then it is possible for particles to be finding their way out of the simulation domain. All particle simulations should include an absorbing box on the perimeter of the simulation. This is automatic in visual setup simulations, and is the responsibility of the user in text setup examples. In some rare cases there is an interaction between a separate absorber on a geometry and the perimiter abosrber that can lead to crashes due to particles getting outside the domain. The simulation is trying to determine a field at a location that is not in memory. This is very rare and should be reported to support@txcorp.com. This can also happen if the particles are being accelerated out of the domain. In these cases, the increase of dump periodicity is useful, but so also is

the *computePtclLimits* analyzer, which will find particles at highest x, lowest x, highest velocity, and report that information in a history. If this shows particles exiting the simulation, then the input file(s) can be adjusted to avoid the possibility that the particles might escape there.

Other situations where a crash may occur later in a simulation include, where a secondary emitter is poorly defined, and it is some time before the shape on which that emission occurs is struck by a particle that can trigger this process.

12.7 Troubleshooting Performance

VSim is designed to optimally use the computational hardware you have available, whether a laptop or a leadership class supercomputing facility.

It achieves this through the use of advanced algorithms, but this does not guarantee any given simulation will run as fast as possible. This document outlines some simple checks which may aid speeding up a simulation.

On the one hand, there are different types of algorithms for field solves, particle movers and monte-carlo, and these may all scale up in slightly different ways, and require different amounts of inter- processor communication for large parallel simulations. Having many histories may also impact performance.

Firstly, it helps to understand which parts of the simulation are taking the most time. The best way to do this is to remove elements of the simulation one at a time, and to assess the difference in speed. Having measured performance of field solves, particle pushes (if applicable), monte carlo interactions (if applicable) and history objects, it may be possible to simplify some of these, for example by adjusting the number of physical particles per macroparticle.

In general one need not dump the fields and particles more often than is necessary as this will lead to slow visualisation, and the slowing down of the simulation while the data is written.

12.7.1 Electromagnetic Solves

Electromagnetic solves tend to be bound by memory access and the ability to pass boundary data across the network. As a rule of thumb - performance tends not to increase well when the domain on each processor is smaller than 40x40x40 - but this limit will depend on the relative performance of your network fabric and CPU. Also, cells outside perfect electrical conductor take longer than inside, so it can sometimes be worth adjusting the domain decomposition strategy to ensure the load is balanced equally. Minimize the regions over which any MAL or PML boundary conditions are applied, as these will be comparatively slow compared with a normal cell update.

12.7.2 Electrostatic Solves

Electrostatic solves can be very communication intensive. Consequently, there may be a 'sweet spot' in terms of not only the total number of processors to use, but the number of processors per node. System administrators may be able to provide diagnostic tools to help monitor this behavior. With fixed number of nodes, varying the domain size may also help with understanding the performance bottlenecks.

12.7.3 Particle Balancing

Particles may not be evenly distributed throughout the simulation domain for all times, and their distributions may vary from species to species. The 'binning' tool in VSimComposer may be used to count the number of particles vs x,y,z, and modifying the domain decomposition will sometimes help ensure the load is balanced optimally. It can help to temporarily add extra diagnostics to measure total populations of macroparticles for all species as they vary through your simulation, and to focus on those with the most macroparticles.

If the number of macroparticles of some species is changing by many orders of magnitude, consider a strategy which includes macroparticle combining or splitting. See: VSim Reference: Monte Carlo Interactions: NullInteraction Kinds

For densities that are uniformly high, and with consequent low mean free path, consider whether a fluid/hybrid approach might be better.

12.7.4 Particle Interactions

For collisional plasmas, the time to run discharge simulations is often dominated by the computation of interactions between the particles.

The most important thing is to make sure you are using the right set of interactions to describe your plasma. If you are uncertain, consider running a global model to analyze the reaction paths and ensure insignificant paths are eliminated from the simulation.

Monte-carlo type algorithms for interactions should be set such that the timestep is small enough that interaction probabilities are always small. *Resolution Issues with Reactions*, provides a more detailed physical explanation for the resolution issues that may occur. .. as advised in ref:monte_carlo_interactions_package .

12.7.5 Histories

Histories store their data in RAM in between data dumps and can write very large datasets. Some histories need to do non-trivial amounts of computation each time step.

12.7.6 Configuration Issues

The installer Tech-X provides can be expected to work well out of the box on desktop and high performance computing systems.

HPC systems often have high performance parallel systems. Commonly these are set up differently from your home area, and you will need to ensure that you are running with your data being output to a specific partition. Check the cluster documentation for more information.

HPC sytems are sometimes configured with a different MPI (to VSim's required MPICH MPI) in the environment set up by the queue system. In rare cases the MPI installation provided by VSim can pick up the wrong network card or fail to use the correct infiniband driver (normally where this has been customized heavily on those clusters). This will likely manifest as very poor parallel performance. For example, a simulation using sixteen cores and one node may run much faster than a simulation on thirty-two cores and two nodes (subject to the scaling advice above). In these cases we recommend you contact Tech-X support at support@txcorp.com for advice. Modification of the environment is non-trivial and may have unexpected consequences.

12.8 Troubleshooting MPI failure to start on OSX

On some versions of OSX, users have reported issues with MPI processes failing to start. This problem can be identified as it causes errors of the kind

```
[[20648,0],0] bind() failed on error Address already in use
```

It appears a solution for this is simply to run:

```
export TMPDIR=/tmp
```

Then to run VSimComposer from the same terminal.

12.9 Troubleshooting Windows Permissions

If you see permission errors, try running the command prompt as administrator by right clicking the prompt and clicking "Run as administrator".

12.10 Troubleshooting VSimComposer Visualization

12.10.1 On Windows, VSimComposer Claims No Data Found Even Though it Exists

It is possible you are trying to visualize data from a folder other than a straight forward hard drive on the system. VSim relies on VisIt for visualisation and the following are known to cause problems:

- Simulation directory path on a remote file system

- Simulation path containing international characters

- Simulation path uses the Universal Naming Convention (starts \)

On Windows machines, data inside a directory following Universal Naming Convention (UNC), e.g., \machine\directory\file, cannot be visualized using VisIt, the underlying visualization tool to VSimComposer. UNC style paths are not supported. Instead, map the UNC path to a letter drive on the machine.

12.10.2 Particle Style Does Not Update When Grid Boundary is Plotted

If you are having difficulties updating your particle species in the visualization window, it may be due to the order in which you are adding elements to the visualization.

Particles and grids are both meshes in the visualization tool that VSim uses (VisIt), and only the latest mesh gets updated. This may be why updating the particles does not have the desired effect; the latest mesh is the grid and it is the one being updated. If you want to update particles, you can instead add the grid first and the particles second. This sequence will lead to updating the particle sizes and styles correctly.

12.10.3 Field Plots Not Being Updated

In certain simulations, you may be plotting multiple fields and notice that certain ones advance in time while others never update past time 0.

This is due to the fact that the latter fields, as illustrated by the J field in the *Magnetic Fields of Wire (Text-based setup)* example simulation, are actually static fields that are never updated throughout the simulation run. Thus, when these static fields are examined in the *Visualization* window, only the initial field data at time 0 is displayed because that is the only data generated.

ADVANCED SIMULATION TOPICS

13.1 Running a Parameter Sweep in VSim/Vorpal

One can run a parameter sweep by creating a script (.bat file for Windows or .sh file for Linux/Mac).

After following the instructions in *Running Vorpal from the Command Line*, you should be able to run on the command line. The argument `-iargs` can be used to pass a parameter value to your input file for running.

```
vorpalser -i myFile.pre -iargs FREQUENCY=1.e9
```

If you are content with running one file at a time (eg for relatively quick simulations, or where all simulations scale perfectly onto all the cores you have available), the quickest way to scan through a set of frequencies, for instance, might be to write a script (.bat or .sh) to loop over the values.

13.1.1 Linux/Mac

A basic example to loop through a number of wavelengths (2-10 by steps of 1) in the emPlaneWave example is as follows:

```
for number in `seq 2 1 10` ; do
  vorpalser -i emPlaneWaveT.pre -iargs WAVELENGTHS=$number -o emPlaneWaveT_${number} \
  > emPlaneWaveT_${number}.log
done
```

13.1.2 Windows

A basic example to loop through a number of wavelengths (2-10 by steps of 1) in the emPlaneWave example is as follows:

```
FOR /L %n IN (2,1,10) DO vorpalser.exe -i emPlaneWaveT.pre -iargs WAVELENGTHS=%n \
-o emPlaneWaveT_%n > emPlaneWaveT_%n.log
```

In a batch file, use %%n in the above command.

Additionally you may want to use VSimComposer for visualisation after the simulation is complete, which requires you to have the .pre file in the run directory, and one directory per simulation. The script below was written for a cygwin machine with four cores. One core was kept clear so that the machine did not slow too much. The `wc ...  | cut -f2 -d" "` approach will not work for larger core counts as the format of wc will change if there are more characters. The **awk** command is pretty standard and would be better. This is principally to provide inspiration if you wish to do something more complex. If you have comments and suggestions please email technical support.

13.2 Selecting Solvers and Solver Parameters

Solvers are used when the field is defined implicitly, i.e., when there is a relation between the field values at various locations and what one is given. For example, in an electrostatic simulation, the potential is found by solving Poisson's equation,

$$-\nabla^2 \phi = \rho$$

This partial differential equation is then discretized to obtain a large linear equation that in the problem interior relates a linear combination of the values of the potentials at a node and nearby nodes to the value of the potential at that node. At the boundaries, the value of the potential is directly related to the desired boundary value. Consequently, Vorpal must solve a large linear system, where the number of independent values, i.e., the length of the vector, is the number of field values in the problem.

As an example, in a Poisson solve for a 10 cell by 10 cell problem, because the potential must also be known at the node above the last cell, there are $11 \times 11 = 121$ values of potential to solve for. The matrix for this solve therefore has $121 \times 121 = 14641$ elements. One can see that these matrices become very large as the problem size increases.

Vorpal gives the user a fair number of choices for solving these problems. The first choice is whether to use a direct solver, which uses methods like LU decomposition, or an iterative solver, which finds the solution through successive matrix operations that converge to the solution. This first choice generally depends on the size of the problem. For problems that are too large, one cannot hold the matrix in memory, and so one generally goes with an iterative method. This is not definitive, however, as for smaller problems, an iterative method may still get one to solution more rapidly.

When using a direct solver, it is important to ensure that the matrix is not singular. In almost all cases this is true, but with fully periodic boundary conditions, the Laplace matrix is singular, as can be seen by the fact that the constant function is in its null space. One can make the matrix nonsingular by applying a boundary condition at a single cell. However, this problem has no solution when the total charge in the system does not vanish. Thus, one must ensure that the system is overall neutral. This can be enforced in Vorpal by adding a neutralizing background charge density.

Iterative solvers have no problem with periodic boundary conditions, and with them one need not impose any single-cell boundary condition. Iterative solvers work very well on the Poisson problem. However, for iterative solvers one must choose a tolerance, which is a measure of the residual reduction compared to the initial residual, as the stopping criterion for the iterative solver. Values of 10^{-8} often are sufficient for giving meaningful results. If the solver does not converge, increase the tolerance. If the resulting potentials miss a lot of small scale structure, reduce the tolerance.

For interative solvers, one must also choose a preconditioner. Preconditioners transform the linear system used in the Poisson solver into systems with more favorable convergence behavior. For a simple fully periodic system, a pre-conditioner may not be necessary. For most other cases, using a pre-conditioner significantly improves the convergence behavior. Multi-grid preconditioning tends to yield the best convergence behavior and is especially good for Poisson's equation.

For a comparison of different pre-conditioners and solvers for electrostatic simulations, see P. Messmer et.al. [MB04].

The Visual Setup user interface by default sets up what are believed to be the best solvers, preconditioners, and parameters. However, it also allows the freedom to try others. Experimentation may be needed to arrive at the fastest running simulation.

GLOSSARY

domain The rectangular Cartesian grid. The physical domain is the grid specified by a user. The extended domain is the grid with guard cells added by Vorpal.

extended domain See domain.

FDTD Finite-difference time-domain. The FDTD method is a technique for solving problems in electromagnetics.

float A floating-point number.

guard cell A cell located outside the user-defined simulation grid that Vorpal adds for parallel processing and other computational purposes. Charges cannot be deposited in guard cells, but you can use guard cells when you describe boundary conditions.

HDF5 Hierarchical Data Format Version 5. A library and file format, developed by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign, for storing graphical and numerical data and for transferring that data between computers. Vorpal and VSimComposer output data in hdf5.

input block An input block is an object consisting of parameters. Input blocks can be nested within other input blocks. For example, input blocks for boundary conditions are nested within the input block for an electromagnetic field.

input file A Vorpal simulation file, which has a .pre suffix. Users define a simulation and its variables in an input file. VSimComposer then runs the input file through a preprocessor to produce a processed input file.

MPI Message Passing Interface. An application programming interface (API) for communicating between processes executing in parallel.

multi-grid pre-conditioner A pre-conditioner that enables a solver to use a hierarchy of grids to solve a partial differential equation problem. The multi-grid pre-conditioner applies the results from coarse grids to accelerate the convergence on the finest grid.

parameter A parameter is a variable value (integer, floating-point number, or text string) that users define to create a simulation.

parse To divide input into parts and determine the meaning of each part.

physical domain See domain.

pre-conditioner An algorithm that works with an electrostatic solver to transfer an original linear system matrix into a matrix that has better convergence behavior.

processed input file A Vorpal simulation file, which has a .in suffix. VSimComposer processes the input file to produce a processed input file.

Python An open-source, interpreted scripting language managed by the Python Software Foundation.

SI units The International System of Units (`Le Systeme International d'Unites`), which has seven base units: meter, kilogram, second, ampere, kelvin, mole, and candela.

solver An algorithm that calculates the results of electrostatic problems.

TxPhysics A cross-platform library of computational modules, provided by Tech-X Corporation, for modeling charged particles.

TRADEMARKS AND LICENSING

- Vorpal™ © 1999-2002 University of Colorado. All rights reserved.

- Vorpal™ © 2002-2021 University of Colorado and Tech-X Corporation. All rights reserved.

- VSim™ except for Vorpal™ is © 2012-2021 Tech-X Corporation. All rights reserved.

For VSim™ licensing details please email sales@txcorp.com. All trademarks are the property of their respective owners. Redistribution of any VSim™ input files from the VSim™ installation or the VSim™ document set, including *VSim Installation*, *VSim Examples*, *VSim User Guide*, *VSim Reference*, and *VSim Customization*, is allowed provided that this Copyright statement is also included with the redistribution.

BIBLIOGRAPHY

[VSi]   VSim: an electromagnetics and plasma computational application. https://www.txcorp.com/vsim. Accessed: 2018-08-12.

[BWC11]   Carl A Bauer, Gregory R Werner, and John R Cary. A second-order 3d electromagnetics algorithm for curved interfaces between anisotropic dielectrics on a yee mesh. *Journal of Computational Physics*, 230(5):2060–2075, 2011.

[BL04]   Charles K Birdsall and A Bruce Langdon. *Plasma physics via computer simulation*. CRC press, 2004.

[CFL28]   R Courant, K Friedrichs, and H Lewy. On the partial difference equations op mathematical physics. *Mathematische Annalen*, 1928.

[DM97]   Supriyo Dey and Raj Mittra. A locally conformal finite-difference time-domain (fdtd) algorithm for modeling three-dimensional perfectly conducting objects. *IEEE Microwave and Guided Wave Letters*, 7(9):273–275, 1997.

[MB04]   Peter Messmer and David L Bruhwiler. A parallel electrostatic solver for the vorpal code. *Computer physics communications*, 164(1-3):118–121, 2004.

[NC04]   Chet Nieter and John R Cary. Vorpal: a versatile plasma simulation code. *Journal of Computational Physics*, 196(2):448–473, 2004.

[NCW+09] Chet Nieter, John R Cary, Gregory R Werner, David N Smithe, and Peter H Stoltz. Application of dey–mittra conformal boundary algorithm to 3d electromagnetic modeling. *Journal of Computational Physics*, 228(21):7902–7916, 2009.

[WBC13]   Gregory R Werner, Carl A Bauer, and John R Cary. A more accurate, stable, fdtd algorithm for electromagnetics in anisotropic dielectrics. *Journal of Computational Physics*, 255:436–455, 2013.

[WC07]   Gregory R Werner and John R Cary. A stable fdtd algorithm for non-diagonal, anisotropic dielectrics. *Journal of Computational Physics*, 226(1):1085–1101, 2007.

[Yee66]   Kane Yee. Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media. *IEEE Transactions on antennas and propagation*, 14(3):302–307, 1966.