Guide for 3D WARP simulations of hollow electron beam lenses: Practical explanation on basis of Tevatron electron lens test stand

Vince Moens
vince.moens@epfl.ch
Ecole Polytechnique Fédérale de Lausanne (EPFL)

Supervisor at Fermilab:
Dr. Giulio Stancari
Fermilab, Batavia IL, USA

Supervisor at CERN:
Dr. Stefano Redaelli
CERN, Geneva, Switzerland

Supervisor at EPFL:
Prof. Dr. Leonid Rivkin
EPFL, Lausanne, Switzerland

FERMILAB-TM-2586-APC — June 8, 2014

Acknowledgment

Fermi National Accelerator Laboratory (Fermilab) is operated by Fermi Research Alliance, LLC under Contract DE-AC02-07-CH-11359 with the United States Department of Energy. This research was supported in part by the US DOE LHC Accelerator Research Program (LARP).

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Contents

Chapter 1

Introduction

The purpose of this guide is to help successive students handle WARP. It outlines the installation of WARP on personal computers as well as super-computers and clusters. It furthermore teaches the reader how to handle the WARP environment and run basic scripts. Lastly it outlines how to execute the current Hollow Electron Beam Lens scripts.

This guide is intended for individuals with basic Python literacy and scripting experience. A deep understanding of Python, Fortran or WARP is not required. The aim is to help future students advance the current Hollow Electron Beam Lens simulations using WARP.

The quide will specifically discuss the following points:

- Installation of WARP
- Usage of WARP and other tools
- TEV, vdisk1 and other computing devices
- Explanation of current Hollow Electron Beam Lens (HEBL) script

For a WARP guide, please refer to the guide written by David P. Grote and hsi colleagues [1]. If you wish some more information about the Hollow Electron Beam Lens, refer to the Master Thesis of Vince Moens [2].

Installation documents and guides for WARP can be found on the Electron Lens Wiki page at Fermilab [1].

[1] `https://cdcvs.fnal.gov/redmine/attachments/download/10742/WARP_install.pdf`

Chapter 2

WARP Installation

WARP is compatible with Unix based systems, including Mac OS X. It is not compatible with Windows OS.

2.1 Introduction

For the purpose of this document, TEV refers to the Wilson Cluster at the Fermi National Accelerator Lab in Batavia, Illinois, USA.

A primary set of installation instructions can be found at `warp.lbl.gov` together with the links for the various installation files required. These links will also be listed in this installation guide. A tar-ball has been created secifically for these instructions, which can be found at: `https://cdcvs.fnal.gov/redmine/documents/619`. It is called `WARP_Install`.

In order to easily run and develop simulations, it is recommended to have WARP installed on your local computer as well as on TEV. For smaller simulations that are too heavy for your personal computer, it is recommended to use `vdisk1`.

To access TEV or `vdisk1`, contact the relevant administrators and see that you are given a user account. You can then access it using secure shell (SSH) from anywhere using the following:

```
ssh -Y usaername@tev.fnal.gov
```

where you should replace the user name with your own. The option `-Y` is crucial to enable the export of the graphical X windows.

It is important to note that `vdisk1` is not directly accessible from outside the Fermilab network. If you want to access it from outside the Fermilab network, first use secure shell to access TEV and then `ssh` into `vdisk1` from there, or do the same via the storage server `mrbutts`. Thus to access it, type the following:

```
ssh -Y username@tev.fnal.gov
ssh -Y username@vdisk1
```

While the installation of WARP on personal computers is generally straight forward, Installing WARP on TEV brings a few complications. Nonetheless it is important to notice that on a Mac OS, the installation of Xcode, the Xcode developer tools and XQuartz is necessary. These can be obtained from the website of Apple. On TEV, compatibility issues with the python installed on TEV, compiler issues and super user rights issues can arise. Instead of finding a solution in which one could bind the python package WARP, which has to be installed locally, into the native python of TEV by creating links and changing a lot of PATHS, we decided to just install a clean version of python in the local directory and use that distribution.

These instructions will cover the installation of WARP on personal computers as well as on TEV using parallel processing and single processing.

2.2 Installation procedure

Apparently the most recent version of Mac OS X Maverick, the compiler front end of C, C++ and Objective C/C++, called `clang`*, no longer allows unused arguments when compiling scripts. This causes some issues with the installation of WARP on the newest version of Mac OS X. I have not yet found a solution and will work on this at a later point. Up to now it is recommended to look at the following thread:* `http://stackoverflow.com/questions/22313407/clang-error-unknown-argument-mno-fused-madd-python-package-installation-fa`*.*

Before commencing any installation of WARP, please check that you have all necessary compilers of `gcc`, including `gfortran`. If you do not have these pre-installed, you may obtain them from `gcc.gnu.org`. In 99.9% of the cases, you will already have these pre-installed.

As discussed previously, you should start the installation by downloading the above mentioned tarball `WARP_Install` from the redmine website. Alternatively you may download each package separately from the online repositories, as explained along the installation process. This will ensure that you have the most up to date version of WARP installed, since the tarball might not always be updated by the users of this guide.

In order to download the tarball to your home directory in TEV, first download it to your home directory on your personal computer and then transfer it using `scp`. Type the following into your command line on your personal computer:

```
cd
wget -i https://cdcvs.fnal.gov/redmine/attachments/download/10741/WARP_Install.tar
    .gz
scp -r WARP_Install.tar.gz username@tev.fnal.gov:/home/username/
```

where you should replace `username` with your user name in both instances. For an installation on your local computer, use the same file you just downloaded but omit the copying to TEV.

If you have decided to proceed with the installation via the tarball, you should now unpack it by entering the directory in which you installed it and typing:

```
cd
tar -xvf WARP_Install.tar.gz
```

You will now find a directory labeled `WARP_Install` in your directory.

2.2.1 Installing on a personal computer

Personal computers usually come with Python, Numpy and sometimes even Scipy pre installed. Please check if these are installed. If so you may jump past these parts of the following instructions.

2.2.1.1 Installing Python

The first step is to install Python on your machine. You may also use this to update your installation. WARP is compatible with versions 2 and 3 of Python. Up to now I have always used Python 2.7.6, you may choose to use a different version, but this quide will be written in terms of Python 2.7.6.

First of all enter the `WARP_Install` directory. If you wish to update the version of WARP, execute the following:

```
cd WARP_Install/
wget -i https://www.python.org/ftp/python/2.7.6/Python-2.7.6.tgz
tar -xvf Python-2.7.6.tgz
cd Python-2.7.6
```

`wget` downloads the new Python package and `tar` expands it. It is important that you update the version number in the commands to those of your liking.

Having entered the Python directory, you should prepare the installation by running the `configure` file. You may specify a install location using the `prefix` argument (see installation on TEV). Usually this is not necessary on a personal computer.

```
./configure
sudo make install
```

You can check your python installation by running

```
cd
which python
```

which checks which gives the location of your python installation. For me it is `/usr/bin/python`.

You can furthermore execute Python to check the version number:

```
cd
python
```

which should provide you with an output similar to this:

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

In this case the version number of python is 2.7.6, which was compiled using GCC 4.8.2. You can quit python by typing `quit()`.

2.2.1.2 Installing Numpy

Next you will install the numerical python package.

If you wish to update the numpy package part of the tarball run the following[1]:

```
cd ~/WARP_Install/
git clone http://github.com/numpy/numpy.git numpy
cd numpy
```

else omit the middle command and just enter the `numpy` directory.

Next you should run the setup.py script:

```
sudo python setup.py install
```

The path given to prefix should be the same as that for your python installation. If you haven't given one during the Python installation, don't give one here. Check your numpy installation by loading python and then numpy.

```
cd
python
import numpy
```

If numpy loads without errors, your installation was successfull. In this case your output should look like the following:

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>>
```

You can check your numpy install location by typing `numpy.__path__` after having loaded numpy in python.

[1] This requires git. If you do not have git installed, you may install it through your package manager, for example through the command `sudo apt-get install git`.

2.2.1.3 Installing SciPy

Next you will install the scientific python package.

If you wish to update the scipy package within the tarball run the following[2]:

```
cd ~/WARP_Install/
git clone git://github.com/scipy/scipy.git scipy
cd scipy
```

else omit the middle command and just enter the `scipy` directory within the tarball.

Run the setup.py script by first running build (no prefix) and then install.

```
cd ~/WARP_Install/scipy/
python setup.py build
sudo python setup.py install
```

The path given to prefix should be the same as that for your python installation. If you haven't given one during the Python installation, don't give one here.

Check your scipy installation by loading python and then scipy. Subsequently you should pass the command `scipy.__path__` to python. The path should refer to the python installation in your home directory.

```
cd
python
import scipy
```

If scipy loads without errors, your installation was successful. In this case your output should look like the following:

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import scipy
>>>
```

You can check your scipy install location by typing `scipy.__path__` after having loaded scipy in python.

2.2.1.4 Installing iPython

iPython is a more friendly interface for python. It allows syntax coloring and simplifies certain commands such as `quit()` becomes `quit`. It furthermore allows you to call directories and change paths from inside python in a simple fashion. This package is not required but recommended.

If you wish to update the ipython package within the tarball run the following[3]:

```
cd ~/WARP_Install/
git clone https://github.com/ipython/ipython.git ipython
cd ipython
```

else omit the middle command and just enter the `ipython` directory within the tarball.

Install it using the setup.py file:

```
python setup.py install
```

From now on you may start python by typing `ipython` instead of `python`. The output from starting ipython changes slightly with respect to python and now looks like:

[2]This requires git. If you do not have git installed, you may install it through your package manager, for example through the command `sudo apt-get install git`.

[3]This requires git. If you do not have git installed, you may install it through your package manager, for example through the command `sudo apt-get install git`.

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
Type "copyright", "credits" or "license" for more information.

IPython 1.2.1 --An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

2.2.1.5 Installing Forthon

Having installed python and numpy in your home directory, you now have a fully functioning python distribution for science. We still need the WARP package. Therefore we now install Forthon. This is a binding between Fortran and Python.

If you wish to update the Forthon package within the tarball run the following[4]:

```
cd ~/WARP_Install/
git clone https://github.com/dpgrote/Forthon.git Forthon
cd Forthon
```

else omit the middle command and just enter the `Forthon` directory within the tarball.

Execute the `setup.py` file.

```
cd ~/WARP_Install/Forthon
sudo python setup.py install
```

Upon termination of the script you can check the installation by loading Forthon:

```
cd
ipython
import Forthon
```

If Forthon loads without errors, your installation was successfull. In this case your output should look like the following:

```
In [1]: import Forthon

In [2]:
```

You can check your Forthon install location by typing `Forthon.__path__` after having loaded numpy in python. Additionally you can type `which Forthon` to check for the path of the Forthon package that is used.

2.2.1.6 Installing openmpi

This is only necessary if you do not have a running version of openmpi. If you do, it is not recommended to reinstall. If you decide to reinstall anyways, install it in a different directory than the standard installation.

While this is normally not necessary, some computers or servers might require the installation of openmpi. For example, Mac OS X Maverick requires this installation. We will therefore quickly outline how to install `openmpi`. Openmpi is used for parallel processing in WARP.

First start of by downloading the newest stable release from the Open MPI website. The releases can be found on `http://www.open-mpi.org/software/`. Copy the tar-file to the directory from which you wish to install it, best your WARP install directory. Next you should unpack it using the following command.

[4]This requires git. If you do not have git installed, you may install it through your package manager, for example through the command `sudo apt-get install git`.

```
tar zxvf openmpi-1.8.1.tar.gz
```

You should now have a directory called `openmpi-1.8.1` in your install directory. Enter it and run the configuration file using the following commands:

```
cd openmpi-1.8.1
./configure --prefix=/usr/local
```

This should run without any problems. It checks whether you have all necessary compilers. If not, please install them and run the command again. Next you should compile all binaries and configure the wrappers using `make`. If successful, run the installation:

```
make all
sudo make install
```

2.2.1.7 Installing WARP

This is probably the trickiest part. First you must decide on whether you wish to install a single, a parallel or both versions of WARP. The difference is apparent in how you handle the compilation of `Makefile.Forthon` or `Makefile.Forthon.pympi`. I will first explain the single version and then the parallel version. You may compile both in the same directory. The installations will not interfere with each other. The difference is in how you call warp once it is installed.

In this case you should actually update the package to its newest version before installing. WARP is still in its development process and bugs are therefore constantly being removed. To update run the following:

```
cd ~/WARP_Install/
git clone https://bitbucket.org/berkeleylab/warp.git warp
```

2.2.1.7.1 Single Installation Configuration For a single processor installation, call the warp directory and go into `pywarp90`. Here you will have to make a file called `WarpC.so` which will be placed in the `scripts` directory.

```
cd ~/WARP_Install/warp/pywarp90/
make -f Makefile.Forthon
```

Upon successful compilation, you should find a file called `WarpC.so` in the sister-directory `scripts` which is located in the parent directory. Additionally a folder labeled `build` will appear in your pywarp folder. If you were to reinstall, first delete the `build` directory using `rm -Rf build`.

2.2.1.7.2 Parallel Installation Configuration In order to install a parallel version of WARP, you will have to first find the install directories of openmpi. They are usually located somewhere in `/usr/lib/` and `/usr/include/` or `/usr/local/lib/` and `/usr/local/include/`. You will then have to cd into the `pywarp` directory:

```
cd ~/WARP_Install/warp/pywarp90/
```

Next you will have to adapt the directories in the file `Makefile.Forthon.pympi`. First replace the paths in line 2 of `Makefile.Forthon.pympi`, which defines `FARGS`, with the directories you found before. The header `-I` refers to the `include` directory and `-L` to the `directory`. In my case I adapt the lines to the following:

```
FARGS = --farg "-DMPIPARALLEL -I/usr/lib/openmpi -L/usr/include/openmpi/"
```

Now you should create a new file in `pywarp90` called `setup.local.py`, which reads:

```
if parallel:
  library_dirs = library_dirs + ['/usr/lib/openmpi/']
  libraries = fcompiler.libs + ['mpi','mpi_f77']
```

where the path should be again adapted to your install paths for openmpi.

Once you have modified the first file and created the setup file, or have decided to use the files present in the WARP install directory, you may make `Makefile.Forthon.pympi` using the command:

```
cd ~/WARP_Install/warp/pywarp90
make -f Makefile.Forthon.pympi
```

Upon successful compilation, you should find a file called `WarpCparallel.so` in the directory `scripts` which is located in the parent directory. Additionally a folder labeled `buildparallel` will appear in your pywarp folder. If you wish to reinstall, you should remove this build directory.

For the parallel installation you will also require pyMPI. You may install the newest version by obtaining it from git:

```
cd ~/WARP_Install/
git clone http://portal.nersc.gov/project/warp/git/pyMPI.git
```

or alternatively you may use the version included in the `WARP_Install` directory. You may then install it via:

```
./configure
sudo make install
```

Now call the directory `scripts` and execute the setup.py file.

```
cd ../scripts
sudo python setup.py install
```

The path given to prefix should be the same as that for your python installation. If you haven't given one during the Python installation, don't give one here.

Check your warp installation by loading python and then warp.

```
cd
ipython
import warp
```

If warp loads without errors, your installation was successful. In this case your output should look like the following:

```
In [1]: import warp
# Warp
# Origin date: Mon, 8 Jul 2013 13:06:44 -0700
# Local date: Mon, 8 Jul 2013 13:06:44 -0700
# Commit hash: 4133853
# /usr/local/lib/python2.7/dist-packages/warp/warp.pyc
# /usr/local/lib/python2.7/dist-packages/warp/warpC.so
# Thu May 15 18:01:40 2014
# import warp time 17.6758611202 seconds
# For more help, type warphelp()
```

You can check your warp install location by typing `warp.__path__` after having loaded scipy in python.

2.2.1.8 Installing PyGist

PyGist is the graphical interface of WARP. It is necessary to produce all the visual interpretation of your simulations.

If you wish to update the PyGist package within the tarball run the following[5]:

[5] This requires git. If you do not have git installed, you may install it through your package manager, for example through the command `sudo apt-get install git`.

```
cd ~/WARP_Install/
git clone https://bitbucket.org/dpgrote/pygist.git pygist
cd pygist
```

else omit the middle command and just enter the `pygist` directory within the tarball.

After this you may run the install script as usual.

```
cd ~/WARP_Install/pygist
python setup.py config
sudo python setup.py install
```

You may again check the installation by calling any directory except the install directories, running Python and importing gist. The output should look as follows:

```
In [3]: import gist

In [4]:
```

You can check your pygist install location by typing `gist.__path__` after having loaded numpy in python. Additionally you can type `which gist` to check for the path of the Forthon package that is used.

2.2.2 Installing on TEV or vdisk1

Create a directory where you would want to install Python and WARP. For the purpose of this tutorial we will use the directory `Python` in your home directory.

```
cd
mkdir Python
```

Next we want to make sure that every compilation of a python script from this point is done using the same compilers and your new python installation. There is two methods to do this. The first is the recommended one and automatically loads the new path upon opening a window. In order to do this check for a file named `.bash_profile` in your home directory.

```
cd
ls -lisa
```

If it does not exist, create it by typing:

```
cd
nano .bash_profile
```

Then in the new terminal window insert the following data:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=/usr/local/gcc-4.6.2/bin:/home/USERNAME/Python/bin:$PATH
export PATH

export LD_LIBRARY_PATH=/usr/local/gcc-4.6.2/lib64:/usr/local/gcc-4.6.2/lib
```

Be sure to change the user name on line 10 to your own user name. Close `nano` by typing `Ctrl-O`, `Enter` and `Ctrl-X`.

If it does exist, append the file by opening it in a similar fashion as above using `nano` and append the file with:

```
# User specific environment and startup programs

PATH=/usr/local/gcc-4.6.2/bin:/home/USERNAME/Python/bin:$PATH
export PATH

export LD_LIBRARY_PATH=/usr/local/gcc-4.6.2/lib64:/usr/local/gcc-4.6.2/lib
```

Again, be sure to change the user name on line 3 to your own user name and close `nano` by typing `Ctrl-O`, `Enter` and `Ctrl-X`.

2.2.2.1 Installing Python

We will now install Python 2.7.3 locally into your home directory. Go into `Python-2.7.3` in the directory `WARP_Install`.

```
cd WARP_Install/Python-2.7.3/
```

In order to install python, run the `configure` file, while specifying a prefix. In my case the prefix is `/home/vmoens/Python`.

```
./configure --prefix=/place/where/python/is/to/be/installed
make install
```

You can check whether the directories `bin`, `include`, `lib` and `share` can be found in the directory that you used as prefix above. Furthermore you should run:

```
cd
which python
```

which checks which version of python you use. The output should refer to your install directory, for me it is `/Python/bin/python`.

2.2.2.2 Installing Numpy

Next you will install the numerical python package. In order to install it cd into the directory `numpy` in `WARP_Install` and run the setup.py script.

```
cd ~/WARP_Install/numpy/
python setup.py install --prefix=/place/where/python/is/to/be/installed
```

The path given to prefix should be the same as that for your python installation. Check your numpy installation by loading python and then numpy. Subsequently you should pass the command `numpy.__path__` to python. The path should refer to the python installation in your home directory.

```
cd
python
import numpy
numpy.__path__
```

(do not copy all 4 lines at once into your terminal. it will cause errors.)

I obtained the output: `['/home/vmoens/Python/lib/python2.7/site-packages/numpy']`. You can quit python by typing `quit()`.

2.2.2.3 Installing SciPy

Next you will install the scientific python package. In order to install it cd into the directory `scipy-0.12.0` in `WARP_Install` and run the setup.py script by first running build (no prefix) and then install.

```
cd ~/WARP_Install/scipy/
python setup.py build
python setup.py install --prefix=/place/where/python/is/to/be/installed
```

The path given to prefix should be the same as that for your python installation. Check your scipy installation by loading python and then scipy. Subsequently you should pass the command `scipy.__path__` to python. The path should refer to the python installation in your home directory.

```
cd
python
import scipy
scipy.__path__
```

(do not copy all 4 lines at once into your terminal. it will cause errors.)

I obtained the output: `['/home/vmoens/Python/lib/python2.7/site-packages/scipy']`. You can quit python by typing `quit()`.

2.2.2.4 Installing iPython

iPython is a more friendly interface for python. It allows syntax coloring and simplifies certain commands such as `quit()` becomes `quit`. It furthermore allows you to call directories and change paths from inside python in a simple fashion. This package is not required but recommended. Install it using the setup.py file:

```
$ tar -xzf ipython.tar.gz
$ cd ipython
$ python setup.py install
```

From now on you may start python by typing `ipython` instead of `python`.

2.2.2.5 Installing Forthon

Having installed python and numpy in your home directory, you now have a fully functioning python distribution for science. We still need warp. Therefore we now install Forthon. This is a binding between Fortran and Python.

Start of by going into the Forthon directory and executing the `setup.py` file.

```
cd ~/WARP_Install/Forthon-0.8.11
python setup.py install --prefix=/place/where/python/is/to/be/installed
```

Upon termination of the script you can check the installation by checking the path of the Forthon installation:

```
cd
python
import Forthon
Forthon.__path__
```

The path should again point to your install directory.
For me it points to: `['/home/vmoens/Python/lib/python2.7/site-packages/Forthon']`. Terminate Python with `quit()`. Additionally you can type `which Forthon` to check for the path of the Forthon package that is used.

Given a correct directory for Forthon, you have successfully installed Forthon and can now start installing WARP itself.

2.2.2.6 Installing WARP

This is probably the trickiest part. First you must decide on whether you wish to install a single, a parallel or both versions of WARP. The difference is apparent in how you handle the compilation of `Makefile.Forthon` or `Makefile.Forthon.pympi`. I will first explain the single version and then the parallel version. You may compile both in the same directory. The installations will not interfere with each other. The difference is in how you call warp once it is installed.

2.2.2.6.1 Single Installation Configuration Call the warp directory and go into `pywarp90`. Here you will have to make a file called `WarpC.so` which will be placed in the `scripts` directory.

```
cd ~/WARP_Install/warp/pywarp90/
make -f Makefile.Forthon
```

Upon successful compilation, you should find a file called `WarpC.so` in the directory `scripts` which is located in the parent directory. Additionally a folder labeled `build` will appear in your pywarp folder.

2.2.2.6.2 Parallel Installation Configuration In order to install a parallel version of WARP, you will have to first find the install directories of openmpi. On tev they can be found at `/local/openmpi/`. You will then have to cd into the `pywarp` directory:

```
cd ~/WARP_Install/warp/pywarp90/
```

In the `WARP_Install` directory, which accompanies this script, all necessary modifications have already been made and you may thus skip to the making of `Makefile.Forthon.pympi`.

In case you use an updated version of WARP which you can download from `warp.lbl.gov`, you will have to make these changes again. First replace line 2 of `Makefile.Forthon.pympi`, which defines `FARGS`, with:

```
FARGS = --farg "-DMPIPARALLEL -I/usr/local/openmpi/include -L/usr/local/openmpi/
    lib/"
```

Now you should create a new file in `pywarp90` called `setup.local.py`, which reads:

```
if parallel:
  library_dirs = library_dirs + ['/usr/local/openmpi/lib/']
  libraries = fcompiler.libs + ['mpi','mpi_f77']
```

Once you have modified the first file and created the setup file, or have decided to use the files present in the WARP install directory, you may make `Makefile.Forthon.pympi` using the command:

```
make -f Makefile.Forthon.pympi
```

Upon successful compilation, you should find a file called `WarpCparallel.so` in the directory `scripts` which is located in the parent directory. Additionally a folder labeled `buildparallel` will appear in your pywarp folder.

For the parallel installation you will also require pyMPI. You may install the newest version by obtaining it from git:

```
cd ~/WARP_Install/
git clone http://portal.nersc.gov/project/warp/git/pyMPI.git
```

or alternatively you may use the version included in the `WARP_Install` directory. You may then install it via:

```
./configure --prefix=/place/to/install
make install
```

Now call the directory `scripts` and execute the setup.py file.

```
cd ../scripts
python setup.py install --prefix=/place/where/python/is/to/be/installed
```

again you should use the same prefix directory as for your other installations. You make check the successful installation of WARP by launching Python, loading WARP and checking its path.

```
cd
python
import warp
warp.__path__
```

I obtain the following result `['/home/vmoens/Python/lib/python2.7/site-packages/warp']`. Terminate python using `quit()`.

2.2.2.7 Installing PyGist

Be certain to have the X11 header files installed.

PyGist is the graphical interface of WARP. It is the most tricky to install since it has hard coded directories in the install code that require super user privileges and need to be changed.

Using an editor of your liking you will need to adapt the code. For the purpose of this example I will use VIM, since it is a terminal integrated editor that supports syntax highlighting.

Start of by calling the pygist directory and opening the setup.py file in your editor.

```
cd ~/WARP_Install/pygist
vim setup.py
:set number
:syntax on
```

Now go to line 445 and replace `'/home/vmoens/Python/bin'` with the `bin` directory in the directory you have used above for prefix. Do the same on line 452. You may edit in vim by typing `i` and leave the editing mode by pressing `Esc`. You may go to a line by typing `:` and then the number. For example `:445`.

After having changed the lines, leave the editor (type `:wq`) and configure the install script. After this you may run the install script as usual.

```
cd ~/WARP_Install/pygist
python setup.py config
python setup.py install --prefix=/place/where/python/is/to/be/installed
```

You may again check the installation by calling any directory except the install directories, running Python and importing gist. Now check the path in which gist is installed.

```
cd
python
import gist
gist.__path__
```

I obtain `['/home/vmoens/Python/lib/python2.7/site-packages/gist']`. You may again quit python by typing `quit()`.

You have now successfully installed WARP on TEV. If you wish you may now delete that `WARP_Install` directory and its tarball.

2.3 Reinstall & Update

Updates and reinstalls are easily possible. In order to reinstall a part of the software from a install directory that you have already used, you will have to ensure that all folders labeled `build` and

`buildparallel` have been removed from your installation directory. Once this is completed, you may proceed as you did during the installation above. Updates versions of all the packages are either available from `warp.lbl.gov` or from the respective python repositories. As a rule of thumb, you may always just reinstall what is in a subsection in the instructions above. The different subsections in the installation instructions should be independent of each other. Only for the installation of the `warp` directory, will you have to go through the compilation of the makefiles and the installation of the scripts directory.

Chapter 3

Using iPython, WARP, Gist, MPIRUN, PyMPI, qsub, qstat

Before we discuss the execution of WARP scripts on the various devices referenced in this guide, we will shortly introduce each of the packages that we have just installed and you will interact with. We will start of with iPython, the more user friendly front end for python. Next we'll discuss WARP, Gist and then the tools needed for parallel processing, `mpirun` and `PyMPI`.

3.1 Using iPython

IPython is focused on facilitating interactive computing in any language. It has its own kernel that interprets your commands and psses them on to the specific language used. Furthermore it supports notebook documentation and several tools for high performance parallel computing. Up to now we have solely used it to provide a more interactive method to interact with Python, including syntax highlighting, directory manipulation and simplification of certain codes.

After starting IPython with the command `ipython`, syntax highlighting should be immediately visible. The input lines have also changed from the standard `> > >` to the Mathematica style input lines of `In [1]:`.

As mentioned, some commands change, such as `quit()` becomes `quit`. For a nice list of all of the options that IPython offers, type `%quickref` into the IPython interpreter.

3.2 Using WARP

Once you have loaded Python or IPython, you may load WARP. There are two possible methods to do so. Either type `import warp` or `from warp import *`. The first is preferred since it will give you some background information concerning the warp you are running. If you use that method to load warp, your output should have the following format:

```
In [5]: import warp
# Warp
# Origin date: Mon, 8 Jul 2013 13:06:44 -0700
# Local date: Mon, 8 Jul 2013 13:06:44 -0700
# Commit hash: 4133853
# /usr/local/lib/python2.7/dist-packages/warp/warp.pyc
# /usr/local/lib/python2.7/dist-packages/warp/warpC.so
# Fri May 16 11:10:50 2014
# import warp time 1.66094303131 seconds
```

```
# For more help, type warphelp()
```

David's manual refers to starting warp by calling it directly from the terminal. I have not been able to do so yet. In order to execute a warp script, you may thus run the following command:

```
execfile("Filename.py")
```

For the rest, WARP is rather well explained by David P. Grote's manual[1]. It will therefore not be repeated her. A personal recommendation is to append all the WARP code into a single file. The comments in this file are really useful in order to understand some of the syntax of WARP command and to find the commands that will do what you want.

3.3 Using Gist

Gist is the plotting package used by WARP. When you run a simulation in warp and enable graphical output, the simulation produces a `.cgm` file. `CGM` stands from Computer Graphics Metafile. Alternatively a PostScript file can be produced by setting `makepsfile` to true when setting up the graphical output in WARP. This can be done with the following line in WARP:

```
setup(makepsfile=0)
```

It is not recommended to set the graphical output to Postscript, since this creates much bigger files. If the output is kept to CGM, as suggested, the files must be read via `gist`. In order to open the cgm files, you must leave WARP and from the command line open the cgm files using:

```
gist filename.cgm
```

This should create a similar ouput to the following:

```
moensv@moensv-desktop:~/Dropbox/Fermilab/Results/130731$ gist
    tbench_1307310432_gun_B4V8.000.cgm
tbench_1307310432_gun_B4V8.000.cgm metafile description:
Wed Jul 31 04:32:12 2013; For: vmoens
gist>
```

At the same time it opens up a X-window which shows the results of the simulation. The first page only contains the warp parameters that you also obtain when running `import warp`. You may more forward and backwards in the CGM file by pressing `f` and `b`. You may also go directly to given pages by typing the page number and then `Enter`. You can leave gist by either pressing `q` when on the X-window or `quit` when focused on the terminal.

Alternatively you may also call gist without a specific file from the terminal by typing `gist`. This opens an empty x-window and the gist command line. In the command line you have the following command options:

- **cgm** - cgm command syntax: `cgm cgmout [size]`. Opens a CGM file cgmout for output. The size (default 1000000) is the maximum size of a single file in the output family, in bytes. Subsequent send commands will write to cgmout, unless the send to list is modified (see send).
- **display** - display command syntax: `display host:server.screen [dpi]`. Connects to the specified X server. Subsequent draw commands will write to server, unless the draw to list is modified (see draw). If specified, 40<=dpi<=200 (default 100).
- **draw** - draw command syntax: `draw [page list]`. Copy the page(s) (default current page) from the current CGM input to all display output devices. By default, these are all X windows. Use alternate syntax: `draw to [device#1 ...]` to specify a particular list of devices to be used by the draw command. Without any device numbers, draw to restores the default list of devices. (Use the info command to describe current device numbers.) Page list syntax: `group1 [group2 ...]`. Page group syntax: `n` just page n, `m-n` pages n thru m, `m-n-s` pages n thru m, step s.
- **eps** - eps command syntax: `eps epsout`. Open an Encapsulated PostScript file epsout, write the current page to it, then close epsout. (Note that an EPS file can have only a single page.)

- **free** - free command syntax: `free [device# ...]`. Finish and close the device#(s). If none given, frees all send devices, (Use the info command to describe current device numbers.)
- **help** - This command explains specific syntax, for example `help cgm` describes the syntax of the `cgm` command.
- **info** - info command syntax: `info`. Print descriptions of all current output files.
- **open** - open command syntax: `open cgminput`. Closes the current CGM input file, then opens cgminput. Only a Gist-compliant binary CGM file is legal. The cgminput may be the first file of a family. Subsequent page numbers refer to this input file.
- **ps** - ps command syntax: `ps psout`. Opens a PostScript file psout for output. Subsequent send commands will write to psout, unless the send to list is modified (see send).
- **quit** - This command also has the synonyms **exit** and **end**.
- **send** - send command syntax: `send [page list]`. Copy the page(s) (default current page) from the current CGM input to all display output devices. By default, these are all X windows. Use alternate syntax: `send to [device#1] ....` to specify a particular list of devices to be used by the send command. Without any device numbers, send to restores the default list of devices. (Use the info command to describe current device numbers.) Page list syntax: `group1 [group2 ...]`. Page group syntax: `n` just page n, `m-n` pages n thru m, `m-n-s` pages n thru m, step s.

3.4 Using mpirun and pyMPI

`pyMPI` allows the execution of python scripts in parallel mode. It is required on both, `TEV` and on your personal computer. When running a script (textttScript.py) on a local computer, you should enter the following command:

```
mpirun -np NUMBEROFCORES pyMPI Script.py
```

`mpirun` starts the parallel computation on `NUMBEROFCODES` amount of processors. You should see that this number is less or equal to the number of cores available to your machine. It then starts the parallel python interpreter `pyMPI` and runs the python script `Script.py` in it. This interpreter is not interactive, you will still see the output of the simulation in your terminal.

On TEV, parallel executions are also made using pyMPI, but a run-file must be provided for `qsub`. The general layout of the run-file is the following:

Listing 3.1: elens_complex_P9000VB0p3-5-0p3T.run

```
# execute with: qsub -l nodes=23:amd32 -q amd32 Script.run -A uslarp
#!/bin/bash
#PBS -A uslarp
#PBS -l nodes=23,walltime=24:00:00
cd /fast/uslarp/vmoens/Scripts
/usr/local/openmpi/bin/mpiexec -npernode PROCPERNODE -np NUMBEROFPROCESSES pyMPI Script.py
echo
exit
```

⇒ This line gives the command that need to pass to the terminal for the cution. It is explained later on.

⇒ This sets the account under which the pt is to be run.

⇒ Defines number of nodes and the l time that the script requires on V. See that the walltime is set ger than you actually need. When script terminates, your spot on V is automatically terminated. The ltime provides a security in case your pt hangs up. After this time, your ulation will be killed.

⇒ Calling the Scripts directory.

⇒ mpiexec is used to execute the python ulation on the number of nodes given ve, with PROCPERNODE amount of cesses per node (keep this low, so that have maximal amoutn of memory lable) and a total amount of processes n by NUMBEROFPROCESSES.

Exits the simulation when the script is complete.

3.5 Using qsub

As mentioned in the previous section, in order to run a simulation on `TEV`, scripts must be submitted to the `qsub`-routine via a `run`-file. IT is not intended that scripts are run in your home directory on TEV as you would do on your personal computer. The `run`-file is explained above. In order to send it to qsub, copy the first line as of `qsub` to your terminal. For instance you would submit:

```
qsub -l nodes=23:amd32 -q amd32 Script.run -A uslarp
```

This submits the python script referenced in the run file `Script.run` to qsub using 23 amd32 nodes under the account of uslarp. For an explanation of amd32 and intel12 nodes, visit `www.tev.fnal.gov`.

It is also possible to reserve a few nodes for interactive use on TEV. You should only do this for development purposes and not for running standard python scripts. In this mode, if you connection to TEV cancels, your script aborts. To run an interactive session, enter the following into your Terminal on TEV:

```
qsub -l nodes=1 -q amd32 -A uslarp -I
```

In this case a single amd32 node is reserved on the account of uslarp in interactive mode.

3.6 Using qstat

You can use qstat to check the que for TEV. When you type `qstat` into the a TEV terminal you obtain an output such as the following:

```
[vmoens@tev ~]$ qstat
Job id                  Name            User           Time Use S Queue
------------------------------------------------------------------------
75858.tev                FILENAME       USERNAME       00:00:00 R long_phi
```

The first entry is the job id, it is a specific number that designates your run. NAME is the usually is set to the filename of the script you are trying to run. USER is your user name on TEV. Time Use indicates the total amount of time that the script has been running. Keep in mind that if you use two cores, your time runs twice as fast, since each core counts separately. S signalizes the status of the run. It will start with the letter `Q` indicating that your run is being queued and switch to `R` when enough cores are free and you are next in line. The more cores you ask for your run, the longer the wait will take since priorities are assigned according to number of cores. The column `Queue` indicates which nodes you are targeting. For us, this should just be `amd32` or intel12.

3.7 Checking the progress of a simulation and killing a job.

By appending `-n` to the `qstat` command, you may figure out on which nodes the script is running. This allows you to login to those nodes and check the output of your script. To check the ouput log, check your node name and write down your job-id number. Then login to the specific node (here we use `tev0501` as an example):

```
rlogin tev0501
cd /var/spool/PBS/spool/
cat JOBID.tev.fnal.gov.OU
cat JOBID.tev.fnal.gov.ER
```

You should use the first node that is listed when you run the command `qstat -n`. The first cat command prints the run output to your command line. The second cat command prints the error log to your command line.

In case a script is not running or you wish to delete it for some other reason, check the job ID and type `qdel JOBID` into your command line.

3.8 Reloading a dumped simulation.

WARP allows for the possibility to dump simulations and reload them later on, so that you don't have to recompute everything. This procedure is called dumping and is started with the command `dump()` in the script you run.

In a dump, each processor creates its own dump file, with a name like `test000250_00001_00004.dump`. The first number is the time step, the second the processor number, and the third the number of processors.

In order to run a restart, you need to rerun everything in your script up to and including the `generate()` command or the package "w3d". Then you are ready to call the restart

When you call restart, you only pass in the first part, for example `restart('test000250')`. The rest of the filename is generated automatically, with each processor reading in the appropriate file. Note that the number of processors running the restart must be the same as the number used when the dump was made.

At this point, your simulation has returned to the point where it was when you dumped it, you may now continue producing plots or runnings steps.

Chapter 4

The TEV and vdisk1 computing devices.

Up to now I have used 3 machines for WARP development. My personal computer, the super-cluster `TEV`, the computer `vdisk1`.

For small personal code developments, you should use your own laptop. The computing power should be sufficient. For larger development or test runs, use `vdisk1`. `vdisk1` is only accessible from inside the Fermilab computing network. If you are thus outside of Fermilab, you will have to ssh into `TEV` and then further tunnel to `vdisk1`.

`vdisk1` has 16 AMD opteron cores at 1.4 GHz with 2MB cache size. It furthermore has a total of 32 GB of memory. With very large time steps, this device can handle full lens simulations. In order to get an account, you should talk to Alexander Valishev.

`TEV` has several possible nodes available. You should use the `intel12` nodes or the `amd32` nodes. The `intel12` nodes consist of 26 dual socket, six core Intel Westmere CPU systems (12 cores/node, 312 cores in total). These nodes deliver a total of 2.37TFlop/s. Each Intel node has 12 GB of 1333MHz DDR3 memory available. The `amd32` nodes refer to 34 quad-socket, eight-core AMD Opteron CPU systems (32 cores/node), providing 6.2 TFlop/s. Each AMD node has 64 GB DDR2 memory.

On `TEV` it is also important to be aware of the various file systems. The systems available are `/usr/local`, `/home`, `/data` and `/fast`.

The folder `/usr/local` contains the common user applications, compiler ans system tools. It is backed up on a daily basis.

The folder `/home` is the user home directory and has a quota limit of 6 GB per user. It is also backed up on a daily basis.

The folder `/data` is a storage area with 30 GB of quota limit. It is a good place to temporarily place your simulation results.

The folder `/fast` is a high throughput scratch space with 30 GB quota limit per project. Project here refers to the `uslarp` project. It has a throughput of 1GB/s in read and 750 MB/s in write. Due to the large memory swaps of the Electron Lens Simulations, it is recommended to use this

space for running simulations. I have created a folder inside of the uslarp project with my own user name to store my results.

Chapter 5

Explanation of current WARP scripts

This chapter explains the Hollow Electron Beam Lens script as it was at the time of drafting this document. It may have been further developed in the meantime.

5.1 Naming of the scripts

I had produced several versions of the same script, with different settings pre-programmed. I will therefore explain the naming of the HEBL scripts here. Before execution, all scripts start with the name `elens_complex`, for example:

```
elens_complex_P9000VB0p3-5-0p3T.py
```

After the initial name, the potential `P` and the magnetic field `B` are appended. The execution of the script automatically copies the script into the results folder, so that you know which script produced what. The name then changes to:

```
TEL2s_1405281111_gun_P9000VB0p3-50p3T.py
```

Here the `elens_complex` is replaced with the name of the lens type `TEL2s`, the time of execution (28 May 2014, 11:11) and the injection type (`gun`). All other result files have the same initial name, with various other file endings such as `.000.cgm` and `.run`. The last is the run file used to execute the script on TEV.

5.2 Explanation of current HEBL script

This section explains the current script for the simulations. The script is shown on the left and the annotations are found in the right column. The arrows signalize the line to which the annotation belongs. Additional vertical space was introduced at some points in the script in order to leave space for the annotations. This should not affect the behavior of the script.

```

# AUTHOR: Vince Moens
# PROJECT: Master Thesis EPFL 2013

# NOTES ON EMITTANCE TYPES
# GUN: The code automatically injects particles from the cathode conductor using Child Langmuir law.
# PROFILE: Profiles measured in the test bench are injected into the lattice. The gun is ommitted fromt the lattice. Injection takes place at the end of that anode.

############################
# >>> Package Loading <<< #
############################

from warp import *
from datetime import *
import numpy as np

#########################
# >>> File Loading <<< #
#########################

# --- Profiles ---#
fns = [
# new profiles acquired with ACL script
  "../../HG1b/Profile/Results/Chart_Colors/HG1b_121218_9p25A_3-3-3kG_500V_51mA_b_57_8102_particles.txt", #0
  "../../HG1b/Profile/Results/Chart_Colors/HG1b_121218_9p25A_3-3-3kG_8kV_2940mA_58_8099_particles.txt", #1

      ...

  "../../HG1b/Profile/Results/Chart_Colors/HG1b_130521_9p25A_06-24-06kG_3kV_924mA_135_8129_particles.txt", #62
  "../../HG1b/Profile/Results/Chart_Colors/HG1b_130521_9p25A_06-24-06kG_4kV_1368mA_136_8192_particles.txt" #63
]

Voltage=[500, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]

```

⇒ Preambulatory statements.

⇒ I use these formats to define sections he script

⇒ Loading of all the packages necessary un the script.

⇒ This list gives the locations of all density profiles that were measured he test stand for injection into the ulations.

⇒ List of possible cathode potentials the simulation. May be adjusted as led.

```
# --- Selecting profile ---#
item = 4                                        ⇒ Selecting the potential from Voltages.
Bmain = 5                                       ⇒ Magnetic field in main solenoid [Tesla]
print("\nMagetic Field in Main Solenoid: %g T" % Bmain.)
Bgun = 0.3                                      ⇒ Magnetic field in gun solenoid [Tesla]
Bcoll = 0.3                                     ⇒ Mag. field in collector solenoid [Tesla]
Bbend = Bcoll/9.53                              ⇒ Mag. field in bend solenoids [Tesla].
Cathode_Potential = -Voltage[item]              ⇒ Setting the cathode potential
print("Cathode Potential: %g V" % Cathode_Potential)
#Current=Current[item]
Current= pow(Voltage[item],1.5)*5.3e-6          ⇒ Setting the gun current according to
print("Current: %g A (approximate, exact value determined by CLL" % Current)   L with perveance from master thesis: ce Moens.
npart= 500*Current/0.06                         ⇒ Setting number of macro-particles. ation through trial and error.
compact_factor = 10                             ⇒ A factor which reduces the number of ulation time steps performed, which sed when choosing the large-timestep icle mover for arbitrarily magneticed ies set later in the script.
file_ending = "P"+str(-Cathode_Potential)+"VB"+str(int(Bgun*10))+"-"+str(int(Bmain*10))+"-"+str(int(Bcoll*10))+"kG"   Defines the file name ending, given by simulation parameters

###################                             ⇒ In this section we define the shape and
# >>> Options <<< #                               ction type of the simulation.
###################

machine_type          = "TEL2s"                 ⇒ Choose the machien type. Options are
print("Machine type is: "+machine_type+" setup")   bench (tbench), TEL setup (TEL)
if ((machine_type != "tbench") and (machine_type != "TEL2") and (machine_type != "TEL2s")):   a straightened version of the TEL Ls)
  print("Wrong machine type!")
  quit()
machine_injtype      = "gun"                    ⇒ Set the injection type: profile or gun.
print("Injection type is: "+machine_injtype+" injection \n")
if (machine_injtype=="gun"):                    ⇒ Sets the injection type: space-charge
  machine_emittype       = 2                      ted (CLL)(2) or constant current (1).
elif (machine_injtype=="profile"):
  machine_emittype       = 1
else:
  print("Wrong machine_injtype!!")
  quit()
```

```
####################
# >>> Headers <<< #
####################

now=datetime.now()
date=now.strftime("%y%m%d")
time=now.strftime("%H%M")

#if not os.path.exists("../Results/"+date+"/"):
#    os.makedirs("../Results/"+date+"/")
#    print("New day folder created \n")


os.system("cp elens_complex_P"+str(-Cathode_Potential)+"VB"+str(int(Bgun*10))+"-"+str(int(Bmain*10))+"-"+str(int(Bcoll*10))+"kG.py ../Results/"+date+"/"+machine_type+"_"+date+time+"_P"+str(-Cathode_Potential)+"VB"+str(int(Bgun*10))+"-"+str(int(Bmain*10))+"-"+str(int(Bcoll*10))+"kG.py")
os.system("cp elens_complex_P"+str(-Cathode_Potential)+"VB"+str(int(Bgun*10))+"-"+str(int(Bmain*10))+"-"+str(int(Bcoll*10))+"kG.run ../Results/"+date+"/"+machine_type+"_"+date+time+"_P"+str(-Cathode_Potential)+"VB"+str(int(Bgun*10))+"-"+str(int(Bmain*10))+"-"+str(int(Bcoll*10))+"kG.run")

top.runid = machine_type+"_"+date+time+"_"+machine_injtype+"_"+file_ending
if machine_type == "tbench":
  top.pline2 = "Electron Lens Test Bench"
else: top.pline2 = "Tevatron Electron Lens 2"
if machine_emittype == 1:
  top.pline1 = "Constant-injection_" + machine_injtype
elif machine_emittype == 2:
  top.pline1 = "Child-Langmuir_" + machine_injtype
else: top.pline1 = "other injection method"
top.runmaker = "V. Moens"
```

⇒ These variables set the current time date used for file saving.

⇒ This was an attempt to have the e folders in which the output is saved erated automatically by the script. It commented because it was causing many problems.

⇒ This copies the python script being cuted to the results folder.

⇒ This copies the run-script used for b to the results folder.

⇒ Setting the filename for the run

⇒ Setting the top line descriptor for run. This appears on the graphical put.

⇒ Setting the seond line description for gist output.It reads the emission type injection type.

⇒ Sets the name of the individual ning the simulation.

```
#####################                          ⇒ This section sets the lengths and
# >>> Variables <<< #                           pe of the various lens elements.
#####################                           asurements were mostly taken from
                                                AutoCAD files. The descriptors will
# --- Machine Parameters ---#                   lescribed very briefly.
machine_zstart    = .0e0                        ⇒ Baginning of Lens [m]
if machine_type == "tbench":                    ⇒ Distance from anode to collector [m]
  machine_syslen = 2.86
elif ((machine_type == "TEL2") or
                  (machine_type == "TEL2s")):
  machine_syslen = 4.68581
machine_zplat     = machine_syslen              ⇒ Postion: diagnostic screen [m]
machine_piperad   = 3*cm                        ⇒ Inner pipe radius [m]
zfinal            = machine_zstart + machine_syslen
                                                ⇒ Pinnhole position [m]

# --- Electron Gun ---#
# - Cathode
Cathode_zstart        = -29.25*mm               ⇒ Start of cathode
Cathode_zend          = 0.0*mm                  ⇒ End of cathode
Cathode_radi          = 6.75*mm                 ⇒ Inner cathode radius
Cathode_rado          = 12.7*mm                 ⇒ Outer cathode radius
Cathode_radcurvb      = 10*mm                   ⇒ Inner radius of curvature
Cahtode_radcurvs      = 0.5*mm                  ⇒ Outer radius of curvature
Cathode_voltage       = Cathode_Potential       ⇒ Cathode voltage [V]
# - Anode
#The values were taken from a drawing printed on
 tabloid paper and a conversion rate of 2.25mm(
 real)/mm(drawing)
Anode_zstart          = 9.48*mm                 ⇒ Start of Anode
Anode_z1              = Anode_zstart + 1.5*mm   ⇒ z1, ..., z5 together with r1, ...,
Anode_z2              = Anode_zstart + 3.5*mm     designate vertices in the z-r plane
Anode_z3              = Anode_z1 + 9*mm           the anode shape. These are turned
Anode_z4              = Anode_z3 + 11.25*mm       3D conductors through surfaces of
Anode_z5              = Anode_z4 + 5.625*mm       olution.
Anode_zend            = Anode_z5 + 58.5*mm      ⇒ End of Anode on z-axis
Anode_ri              = 14.25*mm
Anode_ro              = Anode_ri+5.33*mm
Anode_r1              = Anode_ri
Anode_r2              = Anode_ro
Anode_r3              = Anode_ri
Anode_r4              = Anode_ri + 0.675*mm
Anode_radtipi         = Anode_ri + 1.5*mm
Anode_radtipo         = Anode_ro -3.5*mm
Anode_r5              = Anode_ri + 5.625*mm
Anode_rendi           = Anode_r5                ⇒ Inner radius at anode end
Anode_rendo           = Anode_rendi + 1.35*mm   ⇒ Outer radius at anode end
```

```
Anode_radcurvb        = 3.5*mm                    ⇒ radcurvb and radcurvs designate the
Anode_radcurvs        = -1.5*mm                     er and smaller radius of curvatures
                                                    l to describe the curvature at the end
                                                     beginning of the anode.
Anode_voltage         = 0.0e0                     ⇒ Anode voltage [V]
# - Electrode F                                    ⇒ Inner electrode
ElectrodeF_zstart     = Cathode_zstart
ElectrodeF_zend       = 0.98*mm
ElectrodeF_z1         = ElectrodeF_zend -0.5*mm
ElectrodeF_z2         = ElectrodeF_zend -1.4*mm
ElectrodeF_ri         = 13.1*mm
ElectrodeF_ro         = ElectrodeF_ri + 1.9*mm
ElectrodeF_r1         = ElectrodeF_ri + 0.5*mm
ElectrodeF_radcurvs   = -0.5*mm
ElectrodeF_radcurvb   = 1.4*mm
ElectrodeF_voltage    = Cathode_Potential
# - Electrode C                                    ⇒ Outer electrode
ElectrodeC_zstart     = Cathode_zstart
ElectrodeC_zend       = 1.97*mm
ElectrodeC_ri         = 20.5*mm
ElectrodeC_ro         = 22.0*mm
ElectrodeC_radcurv    = 0.75*mm
ElectrodeC_z1         = ElectrodeC_zend-0.75*mm
ElectrodeC_voltage    = Cathode_Potential
# - Gun drift pipe                                 ⇒ Drift pipe of the lens
Gun_pipe_zstart       = 84.375*mm
Gun_pipe_zend         = 178.875*mm
Gun_pipe_ri           = 36*mm #Should this not be 3
  cm?
Gun_pipe_ro           = 33.75*mm
Gun_pipe_voltage      = 0.0

if machine_type == "tbench":                      ⇒ This sets the test bench setup. It is
  # ----Solenoids ---#                               activated if the lens type is set to
  # - Gun Solenoid                                   nch.
  tbench_solenoid_gun_zstart = -13*cm
  tbench_solenoid_gun_zend = 37*cm
  tbench_solenoid_gun_radi = 28*cm
  tbench_solenoid_gun_rado =
   tbench_solenoid_gun_radi+5.433*cm
  tbench_solenoid_gun_b    = Bgun                  ⇒ Maximum axial B field [T]

  # - Main Solenoid
  tbench_solenoid_main_zstart = 0.60
  tbench_solenoid_main_zend = 2.52
  tbench_solenoid_main_radi = 0.20
  tbench_solenoid_main_rado =
   tbench_solenoid_main_radi+14.48*cm
  tbench_solenoid_main_b   = Bmain                  ⇒ Maximum axial B field [T]
```

```
# - Collector Solenoid
tbench_solenoid_col_zstart = 2.67
tbench_solenoid_col_zend = 3.17
tbench_solenoid_col_radi = 28*cm
tbench_solenoid_col_rado =
 tbench_solenoid_col_radi + 5.433*cm
tbench_solenoid_col_b    = Bcoll              ⇒ Maximum axial B field [T]

# --- Drift Spaces ---#
# - First Drift
tbench_drift1_zstart     = 37*cm
tbench_drift1_zend       = 0.60
tbench_drift1_ap         = machine_piperad    ⇒ Drift pipe aperture [m]
# - Second Drift
tbench_drift2_zstart     = 2.52
tbench_drift2_zend       = 2.67
tbench_drift2_ap         = machine_piperad    ⇒ Drift pipe aperture [m]

elif ((machine_type == "TEL2") or (machine_type⇒ This line sets the Tevatron Electron
== "TEL2s")):                                    s genometry, in case this setup is
 # --- Solenoids ---#                            sen.
 # - Gun Solenoid
 TEL2_solenoid_gun_zstart = -167.1*mm
 TEL2_solenoid_gun_length = 330*mm
 TEL2_solenoid_gun_zend   =
  TEL2_solenoid_gun_zstart+
  TEL2_solenoid_gun_length
 TEL2_solenoid_gun_radi   = 120*mm
 TEL2_solenoid_gun_rado   = 248*mm
 TEL2_solenoid_gun_b      = Bgun               ⇒ Maximum axial B field [T]

 # --- Bend Solenoids ---#
 # - first bend starting from gun
 TEL2_bendsol1_gun_zstart  = Cathode_zend +
  281.6*mm
 TEL2_bendsol1_gun_zend    =
  TEL2_bendsol1_gun_zstart + 90*mm
 TEL2_bendsol1_gun_length  = 90*mm
 TEL2_bendsol1_gun_ri      = 193*mm
 TEL2_bendsol1_gun_ro      = 265*mm
 TEL2_bendsol1_gun_b       = Bbend              ⇒ Maximum axial B field [T]



 # - first bend starting from gun
 TEL2_bendsol2_gun_zstart  =
  TEL2_bendsol1_gun_zend + 52.9*mm
 TEL2_bendsol2_gun_zend    =
  TEL2_bendsol2_gun_zstart + 90*mm
```

```
TEL2_bendsol2_gun_length  = 90*mm
TEL2_bendsol2_gun_ri      = 193*mm
TEL2_bendsol2_gun_ro      = 265*mm
TEL2_bendsol2_gun_b       = Bbend              ⇒ Maximum axial B field [T]

# - first bend starting from gun
TEL2_bendsol3_gun_zstart  =
 TEL2_bendsol2_gun_zend + 52.9*mm
TEL2_bendsol3_gun_zend    =
 TEL2_bendsol3_gun_zstart + 90*mm
TEL2_bendsol3_gun_length  = 90*mm
TEL2_bendsol3_gun_ri      = 193*mm
TEL2_bendsol3_gun_ro      = 265*mm
TEL2_bendsol3_gun_b       = Bbend              ⇒ Maximum axial B field [T]

# - Main Solenoid
TEL2_solenoid_main_zstart =
 TEL2_bendsol3_gun_zend + 82.8*mm
TEL2_solenoid_main_length = 2688.5*mm
TEL2_solenoid_main_zend   =
 TEL2_solenoid_main_zstart+
 TEL2_solenoid_main_length
TEL2_solenoid_main_radi   = 42.75*mm
TEL2_solenoid_main_rado   = 241*mm
TEL2_solenoid_main_b      = Bmain              ⇒ Maximum axial B field [T]

# ---Bend Solenoids ---#          ### ---These
 are in a linear alignment. We need to put them
 in a bent alignment
# - first bend starting from gun
TEL2_bendsol1_col_zstart  =
 TEL2_solenoid_main_zend + 82.8*mm
TEL2_bendsol1_col_zend    =
 TEL2_bendsol1_col_zstart + 90*mm
TEL2_bendsol1_col_length  = 90*mm
TEL2_bendsol1_col_ri      = 193*mm
TEL2_bendsol1_col_ro      = 265*mm
TEL2_bendsol1_col_b       = Bbend              ⇒ Maximum axial B field [T]

# - first bend starting from gun
TEL2_bendsol2_col_zstart  =
 TEL2_bendsol1_col_zend + 52.9*mm
TEL2_bendsol2_col_zend    =
 TEL2_bendsol2_col_zstart + 90*mm
TEL2_bendsol2_col_length  = 90*mm
TEL2_bendsol2_col_ri      = 193*mm
TEL2_bendsol2_col_ro      = 265*mm
TEL2_bendsol2_col_b       = Bbend              ⇒ Maximum axial B field [T]
```

```
 # - first bend starting from gun
 TEL2_bendsol3_col_zstart  =
  TEL2_bendsol2_col_zend + 52.9*mm
 TEL2_bendsol3_col_zend    =
  TEL2_bendsol3_col_zstart + 90*mm
 TEL2_bendsol3_col_length  = 90*mm
 TEL2_bendsol3_col_ri      = 193*mm
 TEL2_bendsol3_col_ro      = 265*mm
 TEL2_bendsol3_col_b       = Bbend              ⇒ Maximum axial B field [T]

  # - Col Solenoid
 TEL2_solenoid_col_zstart  =
  TEL2_solenoid_main_zend + 548.26*mm
 TEL2_solenoid_col_length  = 345*mm
 TEL2_solenoid_col_zend    =
  TEL2_solenoid_col_zstart+
  TEL2_solenoid_col_length
 TEL2_solenoid_col_radi    = 120*mm
 TEL2_solenoid_col_rado    = 248*mm
 TEL2_solenoid_col_b       = Bcoll              ⇒ Maximum axial B field [T]

# --- Beam size & position ---#
beama0            = 17.5e0*mm [0cm]             ⇒ Beam size in X [m]
beamb0            = 17.5e0*mm [0cm]             ⇒ Beam size in Y [m]
beamap0           = .0e0*mm [0cm]               ⇒ Beam divergence in X [m_x/m_z]
beambp0           = .0e0*mm                     ⇒ Beam divergence in Y [m_x/m_z]
beamx0            = .0e0*mm                     ⇒ Beam centroid in X [m]
beamy0            = .0e0*mm                     ⇒ Beam centroid in Y [m]
beamxp0           = .0e0*mm                     ⇒ Beam centroid velocity in X [m/s]
beamyp0           = .0e0*mm                     ⇒ Beam centroid velocity in Y [m/s]

# --- Beam inject parameters ---#
beamxinject       = .0e0*mm                     ⇒ Injected beam centroid in X [m]
beamyinject       = .0e0*mm                     ⇒ Injected beam centroid in Y [m]
beamxpinject      = .0e0*mm                     ⇒ Injected beam cent. x-velocity [m/s]
beamypinject      = .0e0*mm                     ⇒ Injected beam cent. y-velocity [m/s]
beamainject       = 17.5*mm                     ⇒ Injected beam radius in X [m]
beambinject       = 17.5*mm                     ⇒ Injected beam radius in Y [m]
beamapinject      = .0e0*mm                     ⇒ Injected beam divergence in X [m]
beambpinject      = .0e0*mm                     ⇒ Injected beam divergence in Y [m]
beamainjmin       = 6.75*mm                     ⇒ Injected beam inner radius in X [m]
beambinjmin       = 6.75*mm                     ⇒ Injected beam inner radius in Y [m]
beamzinject       = machine_zstart              ⇒ Beam injection z-position [m]
```

```
###################
# >>> Script <<< #
###################

#-----------------------------#
#     Invoke setup routine   #
#-----------------------------#

setup(makepsfile=0)


winon()


palette("ImageJ_Fire.gp")

top.dipdioset = false

#---------------------------#
#      Particle Loading   #
#---------------------------#

if (machine_injtype == "profile"):
  print("Reading particle positions...")
  posi = fromfile(fns[item], sep=' ')


  npart = len(posi)/2

  posi = reshape(posi, (npart,2))
  print("Calculating charge density according to particle distribution...")
print("Number of macroparticles = %e" % npart)

#----------------------------#
#     Particle Properties   #
#----------------------------#

# --- Particle parameters ---#
electron_Iz   = -Current


cyc_freq      = echarge*Bmain/emass
timestep      = compact_factor*pi/(2*cyc_freq)
electron_vz   = .0e0
electron_ekin = -Cathode_Potential
electron_q    = -1.e0

```

⇒ Up to now we have defined a lot parameters and variables describing electron lens setup. Now the actual putations and WARP codes start.

⇒ `setup` initiates the graphical output. tscript ouput in this case is turned of, asning that `cgm` files will be produced use with gist.

⇒ This turns on the x-window which runs t to the simulation. This is completely ntless when running on TEV.

⇒ This defines the colorpalette that is to used for graphical output.

⇒ Turns of the automatic generation of oles for bends.

⇒ Particles only need to be loaded if the ct type is set to `profile`.

⇒ Imnports particle positions, reading x y positions consecutively, given they separated by a blank space.

⇒ Calculates the number of particles that e been read into the simulation.

⇒ Reorders the input into two columns n x and y positions respectively.

⇒ Beam current [Amps]. Should be pproximation in case of space-charge ssion. WARP will adjust the current.

⇒ Cyclotron frequency: $f_c = \frac{q}{B \times e_m}$

⇒ Timestep size $t = \frac{c_f \times \pi}{2 \times f_c}$

⇒ Beam velocity at emitting surface [$^m/_s$]

⇒ Kinetic energy of electrons [eV]

⇒ Charge state of electrons [ ]

```
vthz          = .0e0                          ⇒ Thermal Velocity of particles [m/s].
                                                ne thermal jitter should be added in
                                                ure simulations.
lrelativity   = true                          ⇒ Whether relativistic effects should be
                                                sidered.
relativity    = true                          ⇒ Level of relativistic correctness (1:
                                                e transverse field by 1/γ²)
sw=int((-electron_Iz*timestep/echarge)/npart) ⇒ Macroparticle weight [ ]
elec = Species(type=Electron,color=red,weight=sw) ⇒ Definition of electrons
#prot = Species(type=Proton,color=green)      ⇒ Definition of protons. Deactivated for
                                                .
elec.ibeam      = electron_Iz                 ⇒ Electron current [A]
print ("Beam Current: %g" % elec.ibeam)
elec.zion       = electron_q                  ⇒ Electron charge state [ ]
print ("Particle charge: %g" % elec.zion)
top.dt          = timestep
print ("Cyclotron Frequency: %g" % cyc_freq)
print ("Timestep: %g" % top.dt)
elec.vbeam      = electron_vz                 ⇒ The electron velocity is set to 0 since it
print ("Particle velocity: %g" % elec.vbeam)    be calculated from the kinetic energy.
elec.ekin       = electron_ekin               ⇒ Kinetic energy in z-direction [eV]
elec.aion       = top.emass/top.amu           ⇒ Electron atomic mass number
top.derivqty()                                ⇒ Turns on the calculation of electron
                                                city from the kinetic energy.
elec.lrelativ   = lrelativity
elec.relativity = relativity
elec.vthz = vthz
#top.vthz = .5e0*top.vbeam*top.emit/sqrt(top.a0*
 top.b0)
#ebeam=(-Cathode_voltage)*echarge+emass*clight**2
#vbeam=clight*numpy.sqrt(1-(emass*clight**2)**2/
 ebeam**2)
nsteps = 1.2*machine_syslen/elec.vbeam/timestep ⇒ Calculated the number of time steps
print("The number of time steps is: %f steps \n" % ded. It is increased by the factor 1.2
  nsteps)                                       pirically to compensate for the reduced
#---------------------#                          tron velocity near the cathode.
#     Beam Design    #
#---------------------#

# - size
elec.a0 = Cathode_rado                        ⇒ Beam size in X
elec.b0 = Cathode_rado                        ⇒ Beam size in Y
elec.ap0 = beamap0                            ⇒ Beam divergance in X
elec.bp0 = beambp0                            ⇒ Beam divergance in Y
# - centroid
elec.x0 = beamx0                              ⇒ Initial beam centroid in x
elec.xp0 = beamxp0                            ⇒ Initial beam centroid in vx/vz
elec.y0 = beamy0                              ⇒ Initial beam centroid in y
elec.yp0 = beamyp0                            ⇒ Initial beam centroid in vy/vz
```

```
#-------------------#
#     Injection     #
#-------------------#

# --- Beam Injection ---#
elec.npmax = npart                                  ⇒ Sets the maximum number of macro-
                                                        icles to be injected.
top.inject = machine_emittype                       ⇒ Defines the type of particle injection.
                                                        urned off
                                                        onstant current
                                                        pace-charge limited (Child-Langmuir)
                                                        pace-charge limited (Gauss's law)
                                                        Richardson-Dushman emission
                                                        nixed Richardson-Dushman and CLL
                                                        ıser specified emission distribution
                                                        Taylor-Langmuir ionic emission
                                                        nixed Taylor-Langmuir and CLL
top.zinject[0] = beamzinject                        ⇒ z-Position of the injection source

# --- Injection Specific Setup ---#

if (machine_injtype=="gun"):                        ⇒ Only when injection is set to "gun":
  elec.npinject =  int(npart**2*sw*elec.sq/(elec⇒ Sets the number of particles injected
   ibeam*timestep*nsteps))                              timestep. npinject=$\frac{n^2 \times m_w \times q_e}{I_e \times t_{del} \times n_{steps}}$
  print("number of particles injected per time
   step: %g" % elec.npinject)

  top.xinject[0] = beamxinject                      ⇒ X location of injection source.
  top.yinject[0] = beamyinject                      ⇒ Y location of injection source.
  top.xpinject[0] = beamxpinject                    ⇒ $v_x/v_z$ of injected particles.
  top.ypinject[0] = beamypinject                    ⇒ $v_y/v_z$ of injected particles.
  top.ainject[0] = elec.a0                          ⇒ Width of injection in x.
  top.binject[0] = elec.b0                          ⇒ Width of injection in y.
  top.ainjmin[0] = Cathode_radi                     ⇒ Minimum of injection in x.
  top.binjmin[0] = Cathode_radi                     ⇒ Minimum of injection in y.
  top.apinject[0] = beamapinject                    ⇒ Convergence angle of injection in x.
  top.bpinject[0] = beambpinject                    ⇒ Convergence angle of injection in y.
  top.vzinject[0,0] = 0.0                           ⇒ Longitudinal velocity at injector
  top.vinject[0] = Cathode_Potential                    cce.
                                                        Sets injector voltage [V]

if machine_injtype == 'profile':                    ⇒ Only when injection is set to "profile":
  xinit = posi[:,0] * mm                            ⇒ Defines x and y Position of injected
  yinit = posi[:,1] * mm                                icles.
  zinit = zeros(npart)                              ⇒ Sets z-Position to 0 for particles.
  vxinit = zeros(npart)                             ⇒ Sets x and y velocity to 0
  vyinit = zeros(npart)
  vzinit = zeros(npart) + elec.vbeam                ⇒ Sets z velocity to the beam velocity.
  def hollow_cathode_source():                      ⇒ Defines the injection of the measured
                                                        m profiles at the cathodes.
```

```
467    if w3d.inj_js == elec.jslist[0]:
468
469      w3d.npgrp = npart
470
471      gchange('Setpwork3d')
472
473
474      w3d.xt[:] = xinit
475      w3d.yt[:] = yinit
476      w3d.zt[:] = top.zinject
477      w3d.uxt[:] = vthz
478      w3d.uyt[:] = vthz
479      w3d.uzt[:] = elec.vbeam
480      # w3d.uzt[:] = top.vbeam
481
482  installuserparticlesinjection(hollow_cathode_source)
483
484 #-----------------#
485 #    Lattice    #
486 #-----------------#
487
488 top.diposet = false
489
490 # The zero point is at the cathode
491 if machine_type == "tbench":
492   # - Gun Solenoid
493   addnewsolenoid(zi=tbench_solenoid_gun_zstart, zf=tbench_solenoid_gun_zend, ri=tbench_solenoid_gun_radi, ro=tbench_solenoid_gun_rado, maxbz=tbench_solenoid_gun_b)
494   # - Drift before main solenoid
495   addnewdrft(zs=0.37, ze=0.60, ap=machine_piperad)
496   # - Main Solenoid
497   addnewsolenoid(zi=tbench_solenoid_main_zstart, zf=tbench_solenoid_main_zend, ri=tbench_solenoid_main_radi, ro=tbench_solenoid_main_rado, maxbz=tbench_solenoid_main_b)
498   # - Drift after main solenoid
499   addnewdrft(zs=2.52, ze=2.67, ap=machine_piperad)
500   # - Collector Solenoid
501   addnewsolenoid(zi=tbench_solenoid_col_zstart, zf=tbench_solenoid_col_zend, ri=tbench_solenoid_col_radi, ro=tbench_solenoid_col_rado, maxbz=tbench_solenoid_col_b)
502
```

⇒ Check whether the electrons are set to njected.

⇒ Defines number of particles to be cted per timestep.

⇒ Changes allocation of dynamic arrays pecified group.

⇒ Allocation of positions defined above he field mesh given by the package

⇒ Installtion of the source "hollow thode_source" into the system.

⇒ Whether to set dipoles in bends omatically.

⇒ These next few lines define the noids and drifts. Inside the brackets, start, end, inner and outer radi are ned. Furthermore the magnetic fields ach solenoid are defined. Bends are ently commented out, because there sists a problem with the beam not perly following the bends around the ner. This is the next thing that should mproved in this code.

```
elif machine_type == "TEL2s":
  # - Gun Solenoid
  addnewsolenoid(zi=TEL2_solenoid_gun_zstart, zf=
   TEL2_solenoid_gun_zend, ri=
   TEL2_solenoid_gun_radi, ro=
   TEL2_solenoid_gun_rado, maxbz=
   TEL2_solenoid_gun_b)
  # - Bend
  #addnewbend(zs=TEL2_solenoid_gun_zend,ze=
   TEL2_solenoid_main_zstart,rc=(
   TEL2_solenoid_main_zstart-
   TEL2_solenoid_gun_zend)/1.02) #Bend angle is
  # - 3 Bends before main solenoid
  addnewsolenoid(zi=TEL2_bendsol1_gun_zstart, zf=
   TEL2_bendsol1_gun_zend, ri=TEL2_bendsol1_gun_ri
   , ro=TEL2_bendsol1_gun_ro, maxbz=
   TEL2_bendsol1_gun_b)
  addnewsolenoid(zi=TEL2_bendsol2_gun_zstart, zf=
   TEL2_bendsol2_gun_zend, ri=TEL2_bendsol2_gun_ri
   , ro=TEL2_bendsol2_gun_ro, maxbz=
   TEL2_bendsol2_gun_b)
  addnewsolenoid(zi=TEL2_bendsol3_gun_zstart, zf=
   TEL2_bendsol3_gun_zend, ri=TEL2_bendsol3_gun_ri
   , ro=TEL2_bendsol3_gun_ro, maxbz=
   TEL2_bendsol3_gun_b)
  # - Main Solenoid
  addnewsolenoid(zi=TEL2_solenoid_main_zstart, zf=
    TEL2_solenoid_main_zend, ri=
   TEL2_solenoid_main_radi, ro=
   TEL2_solenoid_main_rado, maxbz=
   TEL2_solenoid_main_b)
  # - Bend
  #addnewbend(zs=TEL2_solenoid_main_zend, ze=
   TEL2_solenoid_col_zstart,rc=(
   TEL2_solenoid_col_zstart-
   TEL2_solenoid_main_zend)/1.02)
  # - 3 Bends after main solenoid
  addnewsolenoid(zi=TEL2_bendsol1_col_zstart, zf=
   TEL2_bendsol1_col_zend, ri=TEL2_bendsol1_col_ri
   , ro=TEL2_bendsol1_col_ro, maxbz=
   TEL2_bendsol1_col_b)
  addnewsolenoid(zi=TEL2_bendsol2_col_zstart, zf=
   TEL2_bendsol2_col_zend, ri=TEL2_bendsol2_col_ri
   , ro=TEL2_bendsol2_col_ro, maxbz=
   TEL2_bendsol2_col_b)
  addnewsolenoid(zi=TEL2_bendsol3_col_zstart, zf=
   TEL2_bendsol3_col_zend, ri=TEL2_bendsol3_col_ri
   , ro=TEL2_bendsol3_col_ro, maxbz=
   TEL2_bendsol3_col_b)
```

```
  # - Collector Solenoid
  addnewsolenoid(zi=TEL2_solenoid_col_zstart, zf=
   TEL2_solenoid_col_zend, ri=
   TEL2_solenoid_col_radi, ro=
   TEL2_solenoid_col_rado, maxbz=
   TEL2_solenoid_col_b)

elif machine_type == "TEL2":
  # - Gun Solenoid
  addnewsolenoid(zi=TEL2_solenoid_gun_zstart, zf=
   TEL2_solenoid_gun_zend, ri=
   TEL2_solenoid_gun_radi, ro=
   TEL2_solenoid_gun_rado, maxbz=
   TEL2_solenoid_gun_b)
  # - Bend
  addnewbend(zs=TEL2_solenoid_gun_zend,ze=
   TEL2_solenoid_main_zstart,rc=(
   TEL2_solenoid_main_zstart-
   TEL2_solenoid_gun_zend)/1.02) #Bend angle is
  # - 3 Bends before main solenoid
  addnewsolenoid(zi=TEL2_bendsol1_gun_zstart, zf=
   TEL2_bendsol1_gun_zend, ri=TEL2_bendsol1_gun_ri
   , ro=TEL2_bendsol1_gun_ro, maxbz=
   TEL2_bendsol1_gun_b)
  addnewsolenoid(zi=TEL2_bendsol2_gun_zstart, zf=
   TEL2_bendsol2_gun_zend, ri=TEL2_bendsol2_gun_ri
   , ro=TEL2_bendsol2_gun_ro, maxbz=
   TEL2_bendsol2_gun_b)
  addnewsolenoid(zi=TEL2_bendsol3_gun_zstart, zf=
   TEL2_bendsol3_gun_zend, ri=TEL2_bendsol3_gun_ri
   , ro=TEL2_bendsol3_gun_ro, maxbz=
   TEL2_bendsol3_gun_b)
  # - Main Solenoid
  addnewsolenoid(zi=TEL2_solenoid_main_zstart, zf=
    TEL2_solenoid_main_zend, ri=
   TEL2_solenoid_main_radi, ro=
   TEL2_solenoid_main_rado, maxbz=
   TEL2_solenoid_main_b)
  # - Bend
  addnewbend(zs=TEL2_solenoid_main_zend, ze=
   TEL2_solenoid_col_zstart,rc=(
   TEL2_solenoid_col_zstart-
   TEL2_solenoid_main_zend)/1.02)
  # - 3 Bends after main solenoid
  addnewsolenoid(zi=TEL2_bendsol1_col_zstart, zf=
   TEL2_bendsol1_col_zend, ri=TEL2_bendsol1_col_ri
   , ro=TEL2_bendsol1_col_ro, maxbz=
   TEL2_bendsol1_col_b)
```

```
addnewsolenoid(zi=TEL2_bendsol2_col_zstart, zf=TEL2_bendsol2_col_zend, ri=TEL2_bendsol2_col_ri, ro=TEL2_bendsol2_col_ro, maxbz=TEL2_bendsol2_col_b)
addnewsolenoid(zi=TEL2_bendsol3_col_zstart, zf=TEL2_bendsol3_col_zend, ri=TEL2_bendsol3_col_ri, ro=TEL2_bendsol3_col_ro, maxbz=TEL2_bendsol3_col_b)
# - Collector Solenoid
addnewsolenoid(zi=TEL2_solenoid_col_zstart, zf=TEL2_solenoid_col_zend, ri=TEL2_solenoid_col_radi, ro=TEL2_solenoid_col_rado, maxbz=TEL2_solenoid_col_b)

# >>> Set input parameters describing the 3d simulation.
```

⇒ These three lines define the grid on ch the WARP Particle in Cell code ks.

```
w3d.nx = 32
```

⇒ Number of grid points in x

```
w3d.ny = 32
```

⇒ Number of grid points in y

```
w3d.nz = 256
```

⇒ Number of grid points in z

```
top.prwall = machine_piperad
```

⇒ This defines the wall of the simulation, point at which particles are scraped. is set to 0, then the biggest cylinder sed that fits into the simulation mesh.

```
# >>> Set to finite beam.
```

⇒ Sets the dimensions of the mesh in the sical space.

```
w3d.xmmin = -machine_piperad
```

⇒ mesh minimum in x [m]

```
w3d.xmmax = machine_piperad
```

⇒ mesh maximum in x [m]

```
w3d.ymmin = -machine_piperad
```

⇒ mesh minimum in y [m]

```
w3d.ymmax = machine_piperad
```

⇒ mesh maximum in y [m]

```
w3d.zmmin = machine_zstart
```

⇒ mesh minimum in z [m]

```
w3d.zmmax = machine_syslen
```

⇒ mesh maximum in z [m]

```
dx = (w3d.xmmax-w3d.xmmin) / w3d.nx
```

⇒ Grid size in x [m]

```
dy = (w3d.ymmax-w3d.ymmin) / w3d.ny
```

⇒ Grid size in y [m]

```
dz = (w3d.zmmax-w3d.zmmin) / w3d.nz
```

⇒ Grid size in z [m]

```
# >>> Set up some diagnostic windows.
```

⇒ Window 0 is set to full mesh at eration

```
top.xwindows[:,1] = [-5.e-2,5.e-2]
```

⇒ "window" limits for y-z phase space s

```
#top.rwindows[:,1] = [0.e0,.01e0]
```

⇒ radial "window" limits for z-vz phase ce plots

```
top.zwindows[:,1] = [machine_zstart,2*elec.vbeam*top.dt]
```

⇒ "window" limits for x-y phase space s

```
top.zwindows[:,2] = [machine_syslen/2-elec.vbeam*top.dt, machine_syslen/2+elec.vbeam*top.dt]
```

```
top.zwindows[:,3] = [machine_syslen-2*elec.vbeam*
 top.dt, machine_syslen]

# >>> Time histories
elec.nhist = int(nsteps/10)                    ⇒ Interval between timesteps at which
                                                 ory data is saved
top.ifzmmnt = 2                                ⇒ z moments calculation
                                                 one, 1:global moments only, 2:full z
                                                 nents)
top.itmomnts[0:3]=[0,nsteps,elec.nhist]        ⇒ time steps to do calculation of z
                                                 nents and print one-liner of info; first
                                                 e a do loop
top.zmmntmin = machine_zstart                  ⇒ Moments grid minimum in Z
top.zmmntmax = machine_syslen                  ⇒ Moments grid maximum in Z
top.nzmmnt = w3d.nz                            ⇒ Number of points in z moments grid.

#--- Setup Plots                               ⇒ For the following commands, the input
                                                 iven as a loop starting at 0 to nsteps
                                                 the intervall given by the 3^rd input.
                                                 further inputs are specific instances
                                                 o the plots. The input right after the
                                                 command gives the windows in which
                                                 plots should be performed.
top.itplps[0:3]   = [0,nsteps,nsteps]          ⇒ Time steps to do full set of phase space
                                                 s; first 3 are a do loop
# top.itplfreq[0:3] = [0,nsteps,nsteps/20]     ⇒ time steps to do "frequent" phase
                                                 e plots; first 3 are a do loop
top.itplalways[0:3] = [0,nsteps,nsteps]        ⇒ time steps to do "always" plots; first 3
                                                 a do loop
top.itplseldom[0:3] = [0,nsteps,nsteps]        ⇒ time steps to do "seldom" plots; first
                                                 e a do loop
top.pboundnz = absorb                          ⇒ bound. cond. at iz = 0
top.pbound0 = absorb                           ⇒ bound. cond. at iz = nz
top.pboundxy = absorb                          ⇒ boundary conditions
                                                 and y boundaries. absorb/dirichlet:
                                                 orption, reflect/neumann: reflection,
                                                 odic: periodicity

# >>> set up field solver
w3d.solvergeom = w3d.XYZgeom                   ⇒ Defines the geometry of the field
                                                 er. Radial, planar and 3D geometries
                                                 t. You may look them up in the
                                                 RP scripts.
w3d.bound0     = 1                             ⇒ Type of boundary condition at plane
                                                 )
w3d.boundnz    = 1                             ⇒ Type of bound. condition at plane
                                                 z
w3d.boundxy    = 0                             ⇒ Type of bound. condition at sides
                                                 constant potential,
                                                 zero normal derivative,
                                                 2 is periodic
```

```
if w3d.solvergeom == w3d.XYZgeom:
  # >>> Set some flags only needed if using the 3d solver
  w3d.l4symtry   = true                          ⇒ Turns on 4-fold symmetry to simplify
                                                   ulation. Might have to be turned of
                                                   ee full space-charge evolution.
  top.fstype     = 7                             ⇒ Specifies the type of field solver to be
                                                   d.
                                                   none
                                                   ine-sine-periodic FFT (the default),
                                                   8-fold symmetric capacity matrix in

                                                   apacity matrix for quadrupoles,
                                                   not used)
                                                   d sine-sine FFT + tridiag in z,
                                                   eneral capacity matrix in kz space,
                                                   eneral capacity matrix,
                                                   nultigrid solver,
                                                   arallel solver (in development),
                                                   arallel solver (in development),
                                                   RZ multigrid solver,
                                                   Chombo AMR/multigrid solver,
                                                   Use field solver registered in python,
                                                   3d multigrid, Boltzmann electrons
  f3d.mgparam    = 1.2                           ⇒ Acceleration param. for multigrid
                                                   er
  f3d.downpasses = 1                             ⇒ Defines number of downpasses
  f3d.uppasses   = 1                             ⇒ Defines number of uppasses
  f3d.gridmode   = 1                             ⇒ Sets whether grid is fixed or updates
                                                   omatically. In this case it is fixed.
  f3d.mgverbose  = 1                             ⇒ Level of verbosity of multigrid solver
  f3d.mgntverbose = 1                            ⇒ Time step period when convergence
                                                   rmation is printed.
  f3d.lcndbndy   = true                          ⇒ Turns on sub-grid boundaries
  f3d.lprecalccoeffs = true                      ⇒ Precalculates the finite difference
                                                   ficients and saves them on a mesh.
                                                   ter but uses more memory.
  f3d.laddconductor = false                      ⇒ Call python function calladdconductor
                                                   eginning of field solve when true.
top.lgridqnt = true                              ⇒ Ensures that zgrid is always an integer
                                                   ber of dz.
top.lvinject = true                              ⇒ Sets whether source is included in field
                                                   er.
# Setup Envelope Boundaries
env.zl = w3d.zmmin                               ⇒ Starting z for envelope calculation.
env.zu = w3d.zmmax                               ⇒ Ending z for envelope calculation.
env.dzenv = machine_syslen/1000                  ⇒ Envelop step size [m]
```

```
w3d.interpdk[1]=1
```

⇒ Crucial aspect of code. Turns on the ft-Lorentz mover allowing a time step ch is larger than the gyroperiod. This discussed above at the compaction or.

```
w3d.igradb=1
```

⇒ Specifies method of calculating grad B. ooking up in table ssuming quadrupoles ookup in z, quad in x and y.

```
from loadgradb import setbsqgrad
```

⇒ Loads the package that generates the ay of grad B^2 data.

```
setbsqgrad(w3d.nx,w3d.ny,w3d.nz,w3d.xmmin,w3d.xmmax,w3d.ymmin,w3d.ymmax,w3d.zmmin,w3d.zmmax)
```

⇒ Generation of grad B^2 array.

```
# Generate Envelope function
package("env");generate();step
```

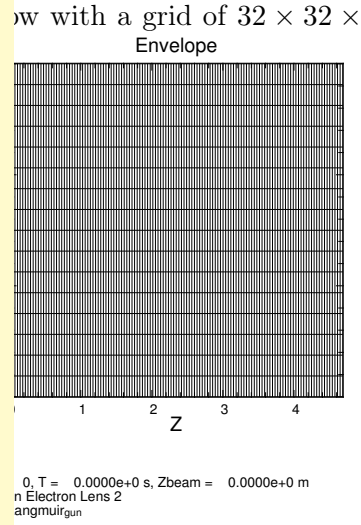
⇒ Loads the elope package, generates the envelope advances the simulations 1 step.

```
# >>> Generate the PIC code (allocate storage, load ptcls, t=0 plots, etc.).
package("w3d");generate()
```

⇒ Loads the field solver package, erates the grid and allocates storage.

```
# >>> Plot the PIC grid
plotgrid()
```

⇒ This command creates a simple 2D of the grid. An example is shown ow with a grid of $32 \times 32 \times 256$ cells.



title envelope is to be neglected e. It is a remnant of another plot that been placed on top of the grid.

```

#-------------------------------#
#     Installing Conductors   #
#-------------------------------#

if (machine_injtype=="gun"):




  # --- Electron Gun ---#
 # - Gun Drift Pipe
  gun_driftpipe=ZCylinderOut(radius=Gun_pipe_ri,
   zlower=Gun_pipe_zstart,zupper=Gun_pipe_zend,
   xcent=0.0,ycent=0.0, voltage=Gun_pipe_voltage)
  # - Cathode
  gun_cathode_r = [Cathode_radi,Cathode_radi,
   Cathode_rado,Cathode_rado]
  gun_cathode_z = [Cathode_zstart,Cathode_zend,
   Cathode_zend,Cathode_zstart]
  gun_cathode_radi=[None,Cathode_radcurvb,None,
   None]
  gun_cathode=ZSrfrv(rsrf=gun_cathode_r,zsrf=
   gun_cathode_z,rad=gun_cathode_radi,voltage=
   Cathode_voltage,xcent=.0e0,ycent=.0e0,zcent=.0
   e0)
  # - Anode
  gun_anode_r         = [Anode_r1,Anode_r3,Anode_r4
   ,Anode_r5,Anode_rendi,Anode_rendo,Anode_rendo,
   Anode_r2,Anode_r2,Anode_radtipo,Anode_radtipi]
  gun_anode_z         = [Anode_z1,Anode_z3,Anode_z4
   ,Anode_z5,Anode_zend,Anode_zend,Anode_z5,
   Anode_z4,Anode_z2,Anode_zstart,Anode_zstart]
  gun_anode_radi      = [None,None,None,None,None,
   None,None,None,Anode_radcurvb,None,
   Anode_radcurvs]
  gun_anode           = ZSrfrv(rsrf=gun_anode_r,
   zsrf=gun_anode_z,rad=gun_anode_radi, voltage=
   Anode_voltage,xcent=.0e0,ycent=.0e0,zcent=.0e0)
  # - Electrode F
  gun_electrodef_r    = [ElectrodeF_r1,
   ElectrodeF_ro,ElectrodeF_ro,ElectrodeF_ri,
   ElectrodeF_ri]
  gun_electrodef_z    = [ElectrodeF_zend,
   ElectrodeF_z2,ElectrodeF_zstart,
   ElectrodeF_zstart,ElectrodeF_z1]
  gun_electrodef_radi = [ElectrodeF_radcurvb,None,
   None,None,ElectrodeF_radcurvs]
```

⇒ This section installs all the conductors n as the cathode and anodes into the em. This is setup dependen and thus nulated into if-loops. The installation one using surfaces of revolution and installed cylinder elements. The faces of revolution require the radial longitudinal positions or the vartices well the as the radius of curvature etween the vartices.

⇒ These are each time created just ore the execution of the Surfaces fo olution command `ZSrfrv`.

```
gun_electrodef      = ZSrfrv(rsrf=gun_electrodef_r,zsrf=gun_electrodef_z,rad=gun_electrodef_radi,voltage=ElectrodeF_voltage,xcent=0,ycent=0,zcent=.0e0)
# - Electrode C
gun_electrodeC_r    = [ElectrodeC_ro,ElectrodeC_ro,ElectrodeC_ri,ElectrodeC_ri]
gun_electrodeC_z    = [ElectrodeC_z1,ElectrodeC_zstart, ElectrodeC_zstart,ElectrodeC_z1]
gun_electrodeC_radi = [None,None,None,ElectrodeC_radcurv]
gun_electrodeC      = ZSrfrv(rsrf=gun_electrodeC_r,zsrf=gun_electrodeC_z,rad=gun_electrodeC_radi,voltage=ElectrodeC_voltage,xcent=0,ycent=0,zcent=.0e0)

gun_conductors=[gun_driftpipe,gun_driftpipe,gun_cathode,gun_anode,gun_electrodef,gun_electrodeC]
installconductor(gun_conductors)
```

⇒ Having piled a list of all the conductors as `_conductors`, this command installs n into the simulation.

```
# ---Lattice ---#
pipe = ZCylinderOut(radius=machine_piperad,zlower=Gun_pipe_zstart,zupper=machine_syslen,voltage=0.,xcent=0,ycent=0,zcent=0)
lattice_conductors=[pipe]
installconductor(lattice_conductors)
```

⇒ Installation of drift pipe throughout lens.

```
fieldsolve()
```

⇒ This tells the interpreter to do a lsol, measning that it calculates the tromagnetic fields in the lens.

```
#-------------------------#
#    Plotting Lattice   #
#-------------------------#
```

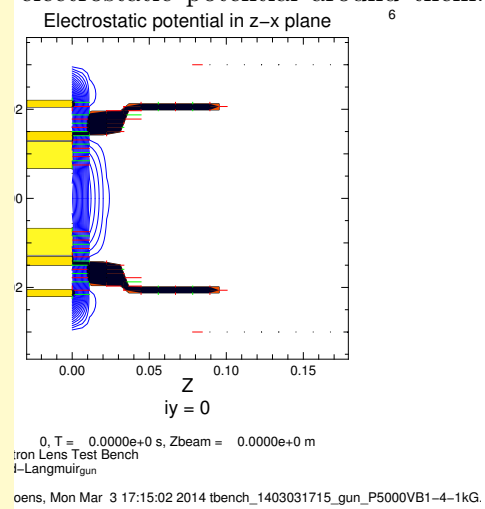
⇒ Having finished the setup of the lens, part sets up the plots.

```
# ---Plotting Envelope Function
penv(color="fg",marks=0,marker=None,msize=1.0,lframe=0,titles=1,ascale=None,bscale=None,zscale=None)
fma()
```

⇒ This is the envelope plot which was laid on the previous grid plot. It sn't produce a proper output which needs to be ivnestigated.

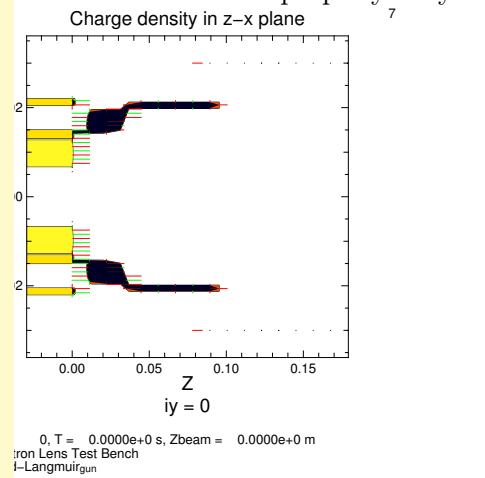
```
# ---Plotting Potential
if (machine_injtype=="gun"):
  gun_driftpipe.draw(filled=190,color="fg")
  gun_cathode.draw(filled=160,color='fg')
  gun_anode.draw(filled=100,color='fg')
  gun_electrodef.draw(filled=150,color='fg')
  gun_electrodeC.draw(filled=150,color='fg')
pfzr(fullplane=1,plotsg=1,cond=1,fill=1,plotphi=1,
 plotrho=0,plotselfe=0,comp='z',titles=1)
limits(Cathode_zstart,Gun_pipe_zend,-1.2*
 machine_piperad,1.2*machine_piperad)
fma()
```

⇒ This section plots the gun conductors electrostatic potential around them.



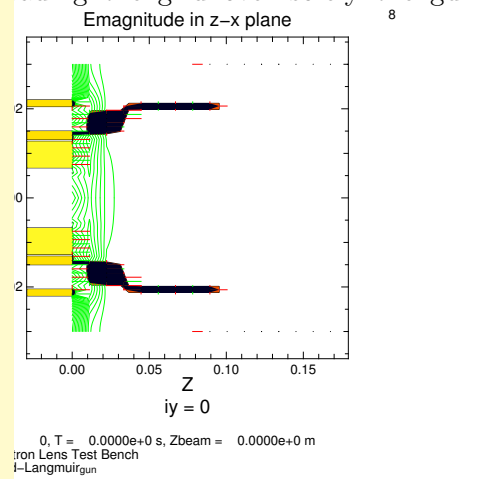

```
# ---Plotting Charge
if (machine_injtype=="gun"):
  gun_driftpipe.draw(filled=190,color="fg")
  gun_cathode.draw(filled=160,color='fg')
  gun_anode.draw(filled=100,color='fg')
  gun_electrodef.draw(filled=150,color='fg')
  gun_electrodeC.draw(filled=150,color='fg')
pfzr(fullplane=1,plotsg=1,cond=1,fill=1,plotphi=0,
 plotrho=1,plotselfe=0,comp='E',titles=1)
limits(Cathode_zstart,Gun_pipe_zend,-1.2*
 machine_piperad,1.2*machine_piperad)
fma()
```

⇒ This section plots the gun conductors the charge density on them. The plot s not work properly yet.



```
# ---Plotting E-field
if (machine_injtype=="gun"):
  gun_driftpipe.draw(filled=190,color="fg")
  gun_cathode.draw(filled=160,color='fg')
  gun_anode.draw(filled=100,color='fg')
  gun_electrodef.draw(filled=150,color='fg')
  gun_electrodeC.draw(filled=150,color='fg')
pfzr(fullplane=1,plotsg=1,cond=1,fill=1,plotphi=0,
 plotrho=0,plotselfe=1,comp='E',titles=1)
limits(Cathode_zstart,Gun_pipe_zend,-1.2*
 machine_piperad,1.2*machine_piperad)
fma()
```

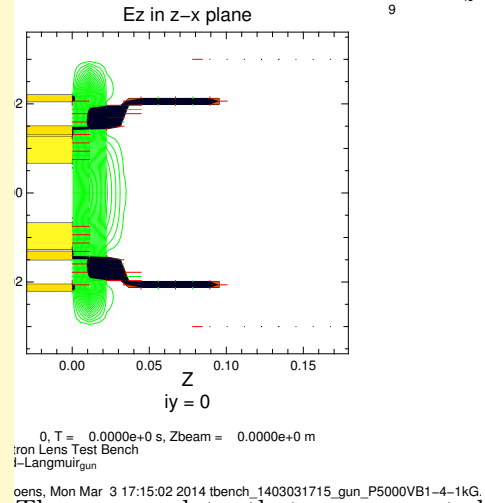
⇒ This section plots the gun conductors the magnitude of the electric field. It ossible to plot specific directions by nging the "comp" command. Better ılts can be obtained by adjusting machine_syslen above and thus eading the grid over solely the gun.



8

```
# ---Plotting Electric Field
if (machine_injtype=="gun"):
  gun_driftpipe.draw(filled=190,color="fg")
  gun_cathode.draw(filled=160,color='fg')
  gun_anode.draw(filled=100,color='fg')
  gun_electrodef.draw(filled=150,color='fg')
  gun_electrodeC.draw(filled=150,color='fg')
pfzr(fullplane=1,plotsg=1,cond=1,fill=1,plotphi=0,
 plotrho=0,plotselfe=1,comp='z',titles=1)
limits(Cathode_zstart,Gun_pipe_zend,-1.2*
 machine_piperad,1.2*machine_piperad)
fma()
## ---REPETITIVE PLOTS
```

⇒ Plots conductors and E_z.

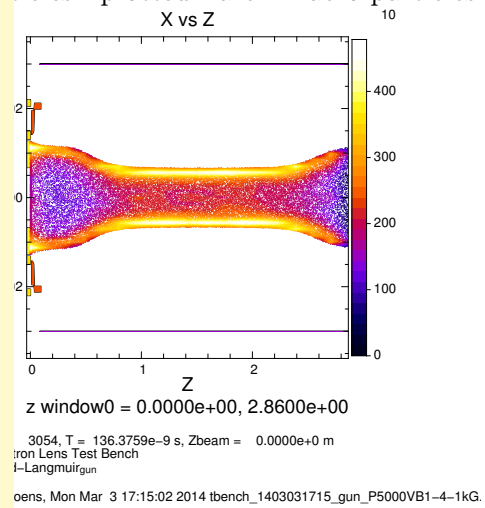


9

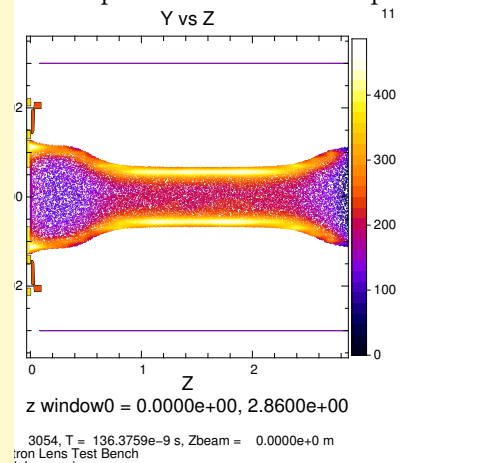
These are plots that are repeated given timesteps. The timesteps are n by the command `itplalways` or `lseldom`.

```python


def myplots():
  # ---Plotting Particles in X-Z plane.
  if (machine_injtype=="gun"):
    gun_driftpipe.draw(filled=190,color="fg")
    gun_cathode.draw(filled=160,color='fg')
    gun_anode.draw(filled=100,color='fg')
    gun_electrodef.draw(filled=150,color='fg')
    gun_electrodeC.draw(filled=150,color='fg')
    pipe.draw(filled=60,color='fg')
  limits(Cathode_zstart,machine_syslen,-1.2*machine_piperad,1.2*machine_piperad)
  # pfzr(fullplane=1,plotsg=1,cond=1,fill=1,plotphi=0,plotrho=0,plotselfe=1,comp='z',titles=1)
  ppzx(color="density",ncolor=30)
  fma()



  # ---Plotting Particles in Y-Z plane.
  if (machine_injtype=="gun"):
    gun_driftpipe.draw(filled=190,color="fg")
    gun_cathode.draw(filled=160,color='fg')
    gun_anode.draw(filled=100,color='fg')
    gun_electrodef.draw(filled=150,color='fg')
    gun_electrodeC.draw(filled=150,color='fg')
    pipe.draw(filled=60,color='fg')
  limits(Cathode_zstart,machine_syslen,-1.2*machine_piperad,1.2*machine_piperad)
  # pfzr(fullplane=1,plotsg=1,cond=1,fill=1,plotphi=0,plotrho=0,plotselfe=1,comp='z',titles=1)
  ppzy(color="density",ncolor=30)
  fma()



  # ---Plots Vz vs. vperp
  ppvzvperp(iw=1,color="density",ncolor=30)
  fma()
```

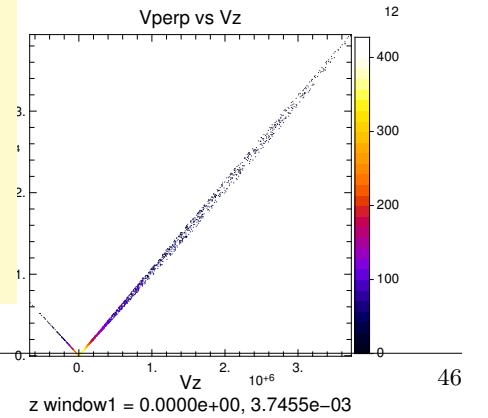
⇒ This plot provides a 2D projection the particles on the X-Z plane. density is color coded. The icles plotted are macro-particles.



⇒ This plot provides a 2D projection the particles on the Y-Z plane. density is color coded. The icles plotted are macro-particles.

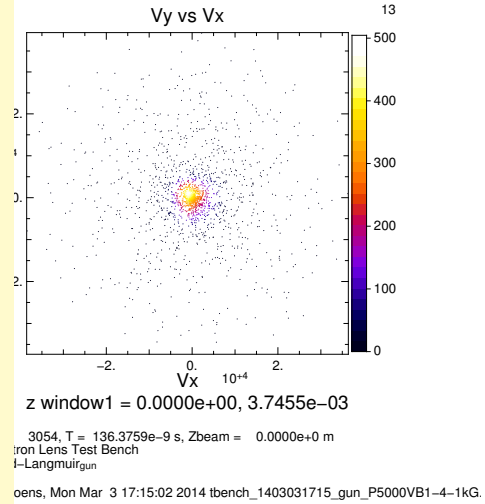


⇒



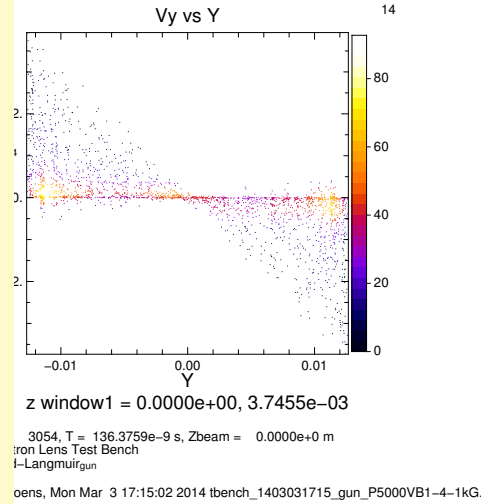
```
# --- Plot vx vs. vy
ppvxvy(iw=1,color="density",ncolor=30)
fma()
```

⇒



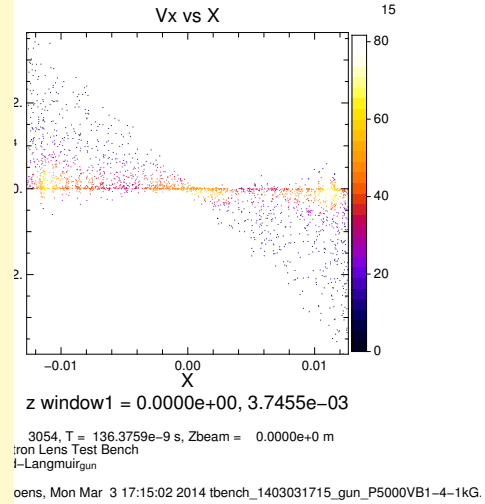

```
# --- Plot y vs. vy
ppyvy(iw=1,color="density",ncolor=30)
fma()
```

⇒



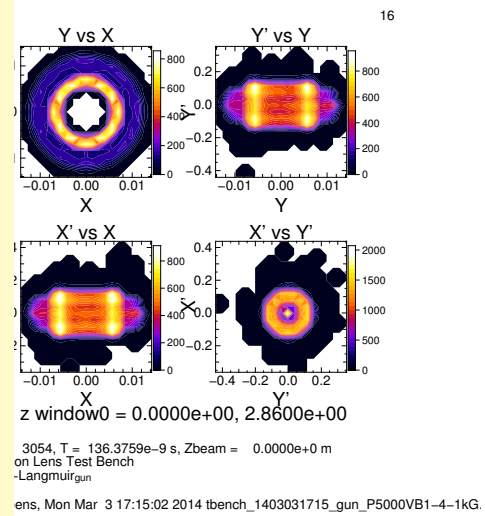

```
# --- Plot x vs. vx
ppxvx(iw=1,color="density",ncolor=30)
fma()
```

⇒



```
# --- Plot phase space
pptrace(filled=1,particles=0,contours=30)
fma()
```

⇒




```
installplalways(myplots)
```

⇒ This installs the plots which were ned above in "myplots".

```
# >>> run time steps and dump the results.
step(nsteps)
```

⇒ This runs the actual particle in codes, running the time steps that ance the particles along the lens. eps defines the number of timesteps are to be run. It was calculated re. For testing purposes this may be aced by a given number.

```
# --- Acquisition of field data
ex=getex()
ey=getey()
ez=getez()
bx=getbx()
by=getby()
bz=getbz()
ke=getke()
```

⇒ Acquisition of electric field data.

⇒ Acquisition of magnetic field data.

⇒ Acquisition of vector potential data.

```
ff = open('../Results/'+date+'/'+machine_type+"_"+date+time+"_"+machine_injtype+"_"+file_ending+'_particlepos.txt','w')
ff.write('#Warp simulation of '+machine_type+'\n# Author: Vince Moens \n# Particle Dump giving position, velocity kinetic energy and the fields at the particles. \n# Date: '+date+'\n# Time: '+time+'\n\n# Time: %10.5e s\n# Number of timesteps: %10.5e \n# Timestep size: %10.5e s\n# Solenoid Fields: %10.5e-%10.5e-%10.5e T\n# Cathode-Anode voltage: %10.5eV\n# Beam current: %10.5e A\n# Beam velocity: %10.5e m/s\n# Bem velocity over c: %10.5e \n# Kinetic Energy: %10.5e eV\n\n' %(top.
```

⇒ The next you lines write the particle itions to a file with the same name but name _particlepos appended. First pens the file using the open command then writes some preambulatory rmation. Next it outputs the itions in 12 columns. First the ition in the first 3 columns, then velocities of the particles, next the etic energy of the particle and lastly electric fields at the particle positions electric and magentic fields in all 3 ctions. It is important to notice that this file gives all the information at the particle positions, not the frid data. The grid data is provided in the next file.

```
dt*int(nsteps),int(nsteps),top.dt,Bgun,Bmain,Bcoll,Cathode_Potential,top.ibeam,top.vbeam,top.vbeamoc,top.ekin))
ff.write('# Number of Macroparticles: %10.5e\n# Macroparticle weight: %10.5e electrons\n# Macroparticle charge: %10.5e coulombs\n\n' %(elec.nps,elec.sw,elec.sq*elec.sw))
ff.write('X[m] Y[m] Z[m] Xv[m/s] Yv[m/s] Zv[m/s] KE[eV] E_x[V/m] E_y[V/m] E_z[V/m] B_x[T] B_y[T] B_z[T] B[T]\n')
for x,y,z,u,v,g,a,b,c,d,e,f in zip(elec.xp,elec.yp,elec.zp,elec.uxp,elec.uyp,elec.uzp,ex,ey,ez,bx,by,bz):
 ff.write('%10.5e %10.5e %10.5e %10.5e %10.5e %10.5e %10.5e %10.5e %10.5e %10.5e %10.5e %10.5e\n' %(x,y,z,u,v,g,a,b,c,d,e,f))
ff.close()

# Obtaining electric fields on grid
allocateselfeforfieldsolve()
nx,ny,nz = array(w3d.phi.shape) #- 1
getselfe3d(w3d.phi,w3d.nxlocal,w3d.nylocal,w3d.nzlocal,
         w3d.nxguardphi,w3d.nyguardphi,w3d.nzguardphi,
         w3d.selfe,w3d.nxguarde,w3d.nyguarde,w3d.nzguarde,
         w3d.dx,w3d.dy,w3d.dz,true)
selfe = w3d.selfe[:,w3d.nxguarde:-w3d.nxguarde or None,
                  w3d.nyguarde:-w3d.nyguarde or None,
                  w3d.nzguarde:-w3d.nzguarde or None]

Ex = selfe[0,...]
Ey = selfe[1,...]
Ez = selfe[2,...]

#Provide fields for whole grid information!

ff = open('../Results/'+date+'/'+machine_type+"_"+date+time+"_"+machine_injtype+"_"+file_ending+'_Efields.txt','w')
ff.write('#Warp simulation of '+machine_type+'\n# Author: Vince Moens \n# Particle Dump \n# Date: '+date+'\n# Time: '+time+'\n\n# Time: %10.5e s\n# Number of timesteps: %10.5e \n# Timestep size: %10.5e s\n# Solenoid Fields: %10.5e-%10.5e-%10.5e
```

⇒ Thsi section obtains the electric fields n the grid data.

⇒ Saving file to file with ending ields. The data is saved in slices ng the z-axis for each direction of E.

```
T\n# Cathode-Anode voltage: %10.5eV\n# Beam current: %10.5e A\n# Beam velocity: %10.5e m/s\n# Bem velocity over c: %10.5e \n# Kinetic Energy: %10.5e eV\n\n' %(top.dt*int(nsteps),int(nsteps),top.dt,Bgun,Bmain,Bcoll,Cathode_Potential,top.ibeam,top.vbeam,top.vbeamoc,top.ekin))
ff.write('# Number of Macroparticles: %10.5e\n# Macroparticle weight: %10.5e electrons\n# Macroparticle charge: %10.5e coulombs\n\n' %(elec.nps,elec.sw,elec.sq*elec.sw))
ff.write('#Grid size in x: %10.5e\n# Grid size in y: %10.5e\n# Grid size in z: %10.5e\n# Cell size in x: %10.5e m\n# Cell size in y: %10.5e m\n# Cell size in z: %10.5e\n' %(w3d.nx,w3d.ny,w3d.nz,dx,dy,dz))
ff.write('\n\nE_x[V/m]:\n\n')
data=Ex
# Write the array to disk. I'm writing a header here just for the sake of readability. Any line starting with "#" will be ignored by numpy. loadtxt
ff.write('# Array shape: {0}\n'.format(data.shape))

# Iterating through a ndimensional array produces slices along the last axis. This is equivalent to data[i,:,:] in this case
for data_slice in data:

  # The formatting string indicates that I'm writing out the values in left-justified columns 7 characters in width with 2 decimal places.
  np.savetxt(ff, data_slice, fmt='%-7.2f')

  # Writing out a break to indicate different slices...
  ff.write('# New slice\n')

ff.write('\n\nE_y[V/m]:\n\n')
data=Ey
# Write the array to disk. I'm writing a header here just for the sake of readability Any line starting with "#" will be ignored by numpy. loadtxt
ff.write('# Array shape: {0}\n'.format(data.shape))

```

```
# Iterating through a ndimensional array produces slices along the last axis. This is equivalent to data[i,:,:] in this case
for data_slice in data:

  # The formatting string indicates that I'm writing out the values in left-justified columns 7 characters in width with 2 decimal places.
  np.savetxt(ff, data_slice, fmt='%-7.2f')

  # Writing out a break to indicate different slices...
  ff.write('# New slice\n')
ff.write('\n\nE_z[V/m]:\n\n')
data=Ez
# Write the array to disk. I'm writing a header here just for the sake of readability Any line starting with "#" will be ignored by numpy. loadtxt
ff.write('# Array shape: {0}\n'.format(data.shape))

# Iterating through a ndimensional array produces slices along the last axis. This is equivalent to data[i,:,:] in this case
for data_slice in data:

  # The formatting string indicates that I'm writing out the values in left-justified columns 7 characters in width with 2 decimal places.
  np.savetxt(ff, data_slice, fmt='%-7.2f')

  # Writing out a break to indicate different slices...
  ff.write('# New slice\n')
# E_y[V/m] E_z[V/m] E[V/m] B_x[T] B_y[T] B_z[T] B[T] Ax[')
# for x,y,z,u,v,g in zip(FIND THIS):
#   ff.write('%10.5e %10.5e %10.5e %10.5e %10.5e %10.5e\n' %(x,y,z,u,v,g))
ff.close()

# Obtaining magnetic fields on grid

bfield = f3d.bfield
nxguardb = bfield.nxguardb
nyguardb = bfield.nyguardb
```

⇒ Obtaining magnetic fields on grid

```
nzguardb = bfield.nzguardb

b = bfield.b[:,nxguardb:-nxguardb or None,
                 nyguardb:-nyguardb or None,
                 nzguardb:-nzguardb or None]
Bx = b[0,...]
By = b[1,...]
Bz = b[2,...]

#Provide fields for whole grid information!
```

⇒ Saving magnetic field data to file in es along the z-axis for each direction he magnetic field.

```
ff = open('../Results/'+date+'/'+machine_type+"_"+date+time+"_"+machine_injtype+"_"+file_ending+'_Bfields.txt','w')
ff.write('#Warp simulation of '+machine_type+'\n# Author: Vince Moens \n# Particle Dump \n# Date: '+date+'\n# Time: '+time+'\n\n# Time: %10.5e s\n# Number of timesteps: %10.5e \n# Timestep size: %10.5e s\n# Solenoid Fields: %10.5e-%10.5e-%10.5e T\n# Cathode-Anode voltage: %10.5eV\n# Beam current: %10.5e A\n# Beam velocity: %10.5e m/s\n# Bem velocity over c: %10.5e \n# Kinetic Energy: %10.5e eV\n\n' %(top.dt*int(nsteps),int(nsteps),top.dt,Bgun,Bmain,Bcoll,Cathode_Potential,top.ibeam,top.vbeam,top.vbeamoc,top.ekin))
ff.write('# Number of Macroparticles: %10.5e\n# Macroparticle weight: %10.5e electrons\n# Macroparticle charge: %10.5e coulombs\n\n' %(elec.nps,elec.sw,elec.sq*elec.sw))
ff.write('#Grid size in x: %10.5e\n# Grid size in y: %10.5e\n# Grid size in z: %10.5e\n# Cell size in x: %10.5e m\n# Cell size in y: %10.5e m\n# Cell size in z: %10.5e\n' %(w3d.nx,w3d.ny,w3d.nz,dx,dy,dz))
ff.write('\n\nB_x[T]:\n\n')
data=Bx
# Write the array to disk. I'm writing a header here just for the sake of readability. Any line starting with "#" will be ignored by numpy. loadtxt
ff.write('# Array shape: {0}\n'.format(data.shape))

# Iterating through a ndimensional array produces slices along the last axis. This is equivalent to data[i,:,:] in this case
for data_slice in data:

```

```
  # The formatting string indicates that I'm writing out the values in left-justified columns 7 characters in width with 2 decimal places.
  np.savetxt(ff, data_slice, fmt='%-7.2f')

  # Writing out a break to indicate different slices...
  ff.write('# New slice\n')

ff.write('\n\nB_y[T]:\n\n')
data=By
# Write the array to disk. I'm writing a header here just for the sake of readability. Any line starting with "#" will be ignored by numpy.loadtxt
ff.write('# Array shape: {0}\n'.format(data.shape))

# Iterating through a ndimensional array produces slices along the last axis. This is equivalent to data[i,:,:] in this case
for data_slice in data:

  # The formatting string indicates that I'm writing out the values in left-justified columns 7 characters in width with 2 decimal places.
  np.savetxt(ff, data_slice, fmt='%-7.2f')

  # Writing out a break to indicate different slices...
  ff.write('# New slice\n')
ff.write('\n\nB_z[T]:\n\n')
data=Bz
# Write the array to disk. I'm writing a header here just for the sake of readability. Any line starting with "#" will be ignored by numpy.loadtxt
ff.write('# Array shape: {0}\n'.format(data.shape))

# Iterating through a ndimensional array produces slices along the last axis. This is equivalent to data[i,:,:] in this case
for data_slice in data:

  # The formatting string indicates that I'm writing out the values in left-justified
```

```
  columns 7 characters in width with 2 decimal places.
 np.savetxt(ff, data_slice, fmt='%-7.2f')

 # Writing out a break to indicate different slices...
 ff.write('# New slice\n')
# E_y[V/m] E_z[V/m] E[V/m] B_x[T] B_y[T] B_z[T] B[T] Ax[')
# for x,y,z,u,v,g in zip(FIND THIS):
#   ff.write('%10.5e %10.5e %10.5e %10.5e %10.5e %10.5e\n' %(x,y,z,u,v,g))
ff.close()

dump() \marginpar{$\Rightarrow$ Transfers all data to the Results directory. } os.system("mv "+machine_type+"_"+date+time+"_"+machine_injtype+"_"+file_ending+"* ../Results/"+date+"/")
```

Having finished the simulations, we ...p the simulation data so that it can ...eloaded. Explain how to reload.

5.3 Example of running the script in the current environment.

First of all I need to authenticate with Kerberos:

```
moensv@moensv-desktop:~$ kinit vmoens
Password for vmoens@FNAL.GOV:
moensv@moensv-desktop:~$
```

As suggested in previous chapters, I have my scripts stored on the `fast` partition of TEV. If I want to run a new file that is currently still stored on my personal computer I will first upload it to my Scripts directoy:

```
cd /Directory/of/my/script
scp -r elens_complex_P9000VB0p3-5-0p3T.py vmoens@tev.fnal.gov:/fast/uslarp/vmoens/Scripts
```

This will upload my file. The output thereof looks as follows:

```
moensv@moensv-desktop:~$ scp -r elens_complex_P9000VB0p3-5-0p3T.py vmoens@tev.fnal.gov:/fast
    /uslarp/vmoens/Scripts
elens_complex_P9000VB0p3-5-0p3T.py        100%  49KB 49.1KB/s  00:00
```

I now log into my account on TEV and access the Scripts directory:

```
moensv@moensv-desktop:~$ ssh -Y vmoens@tev.fnal.gov
Last login: Thu May 29 08:33:34 2014 from 80-218-119-213.dclient.hispeed.ch
Fermi Linux slf51 INSTALL for FermiGenericServer via CDROM on Thu Oct 6 16:05:06 CDT 2011
...
=========================================================================

       For further information please visit http://tev.fnal.gov
                 or email us at tev-admin@fnal.gov

=========================================================================

[vmoens@tev uslarp]$ cd /fast/uslarp/vmoens/Scripts/
[vmoens@tev Scripts]$
```

Next we need to draft a run file as explained above. Open the run file in vim:

```
vim elens_complex_P9000VB0p3-5-0p3T.run
```

And enter the following into the file:

```
# execute with: qsub -l nodes=23:amd32 -q amd32 elens_complex_P9000VB0p3-5-0p3T.run -A
    uslarp

#!/bin/bash
#PBS -A uslarp
#PBS -l nodes=23,walltime=24:00:00

cd /fast/uslarp/vmoens/Scripts

/usr/local/openmpi/bin/mpiexec -npernode 1 -np 23 pyMPI elens_complex_P9000VB0p3-5-0p3T.py

echo
exit
~
~
-- INSERT --                                          9,1          All
```

Save it by writing `:wq`. Now you have all files in your Script directory to run the simulation. You still need to bcreate the Results folder. So call the results folder and create the correct date folder:

```
cd ../Results/
mkdir 140529
cd ../Scripts/
```

You are now ready to run the simulation. Obtain the run code from your run file by copying the first line and submit it with qsub:

```
[vmoens@tev Scripts]$ cat elens_complex_P9000VB0p3-5-0p3T.run
# execute with: qsub -l nodes=23:amd32 -q amd32 elens_complex_P9000VB0p3-5-0p3T.run -A
    uslarp

#!/bin/bash
#PBS -A uslarp
#PBS -l nodes=23,walltime=24:00:00

cd /fast/uslarp/vmoens/Scripts

/usr/local/openmpi/bin/mpiexec -npernode 1 -np 23 pyMPI elens_complex_P9000VB0p3-5-0p3T.py

echo
exit
[vmoens@tev Scripts]$ qsub -l nodes=23:amd32 -q amd32 elens_complex_P9000VB0p3-5-0p3T.run -A
     uslarp
76091.tev.fnal.gov
[vmoens@tev Scripts]$
```

The simulation has now been submitted to qsub. Its job ID is 76091. Check the status with qstat:

```
[vmoens@tev Scripts]$ qstat
Job id                  Name            User           Time Use S Queue
------------------------------------------------------------------------
75858.tev                STDIN           lammel         00:00:00 R long_phi
76090.tev                runTanV2_3.sh   lebrun         21:55:05 R amd32
76091.tev                ...p3-5-0p3T.run vmoens        00:02:10 R amd32
[vmoens@tev Scripts]$
```

It has been running for a total of 2 minutes and 10 seconds at this point. When the simulation is finished it will disappear from the qstat output and appear in the Results folder you created above. You can then look at the results using gist:

```
cd ../Results/140529/
gist FILENAME.cgm
```

What you do with the results is up to you. I recommend not to leave them on the `fast` partition but to move them with `scp` to `vdisk1`.

Chapter 6

Acknowledgements

At the points I would like to thank the following people in helping me troubleshoot the issues with installing WARP.

David P. Grote David is one of the developers of WARP over at Lawrence Berkley National Laboratory. He was incredibly helpfull in debugging all the error codes from TEV and even created an account on TEV for himself in order to help with the installation of WARP.

Eric G. Stern Eric works at the Scientific Computing Division at Fermilab. He helped out with local issues and the debugging of some of the error messages on TEV.

Moses Chung Moses works at the APC Experimental Beam Physics Department and has used WARP for his PhD thesis. He was helpfull in gathering information from friends, former students and his own work in order to help with the installation.

tev-admin@fnal.gov The admin guys of tev were incredibly useful in answering questions concerning the architecture of tev and the packages that are installed.

Alexander Valishev Alexander organized the accounts for David, which ultimately lead to david being able to solve the issue.

Giulio Stancari Giulio was very helpfull by providing his old isntallation files froma previous version that provided some comparisons and giving helpful tips.

For further questions concernign the installation of WARP, please don't hesitate to ask me by sending me an email to vmoens@fnal.gov.

Recommended Literature

[1] David P Grote et al. WARP Manual. *Lawrence Livermore National Laboratory*, 7000:94550–9234, April 2000.

[2] Vince Moens. Experimental and Numerical Studies on the Proposed Application of Hollow Electron Beam Collimation for the LHC at CERN.

[3] D.p. Grote J.-L. Vay R. H.. Cohen, A. Friedman. Large-timestep mover for particle simulations of arbitrarily magnetized species. *Nuclear Instruments & Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors, and Associated Equipment*, pages 52–57, February 2007.