Poisson Solver Code

Craig Lage

August 30, 2017

1 Description

This code is a simple grid-based Poisson's equation solver intended to simulate pixel distortion effects in thick fully-depleted CCD's. The code builds a 3D rectilinear grid to represent a portion of the CCD, assigns the appropriate charge densities and applied potentials, then solves Poisson's equation using multi-grid methods. A 360^3 grid, which is adequate for most purposes, solves in less than one minute on a typical laptop. The code also includes prescriptions to propagate electrons from a given point of creation by an incoming photon down to the point of collection, including both drift and diffusion. Most data is saved as hdf files. The current code is configured to model the ITL STA3800 CCD, but other CCDs can be modeled by editting the configuration file. Plotting routines are available to plot the potentials, E-Fields, pixel shapes, and electron paths. A description of the code, the measurements which were used to validate the code, and some samples of the output are in the file docs/BF_White_Paper_28Sep16.pdf. Below is a basic description of how to install the code and a number of examples.

The code contains many options, and not all combinations have been tested together. If you find a set of options that does not work as you expect, please let me know. However, all of the example configuration files described in the Examples Section below have been tested.

This work is supported by DOE HEP Grant DE-SC0009999.

Installing: Read the Installation Section below.

Running: The basic syntax is:

src/Poisson < configurationfile >

More details are provided in the Examples Section.

Hopefully you find the code useful. Comments and questions are encouraged and should be addressed to: cslage@ucdavis.edu

2 Installation

Dependencies:

There are two dependencies that need to be installed before you can compile the Poisson code:

1. C++ Boost libraries. There are several options for installing these:
   (a) Ubuntu: Install the boost libraries using: sudo apt-get install libboost-all-dev
   (b) Mac OSX: Assuming you are using homebrew, install using: brew install boost

(c) Build them from source. They can be downloaded from: www.boost.org

2. HDF5 libraries. There are several options for installing these:
   (a) Ubuntu: Install the hdf5 libraries using: sudo apt-get install hdf5-tools
   (b) Mac OSX: Assuming you are using homebrew, install using: brew install hdf5
   (c) Build them from source. They can be downloaded from: www.hdfgroup.org/HDF5/release/obtain5.html
3. After installing the above two dependencies, edit the src/Makefile lines BOOST_DIR and HDF5_DIR to point to their locations.
4. In the src directory, type "make". This should build the Poisson code, and create an executable called src/Poisson. Depending on where you have installed the above libraries, you may need to edit your LD_LIBRARY_PATH environment variable so the system can find the appropriate files for linking.
5. I have included in the src directory a file Makefile.nersc that works for me on NERSC Edison.

Running the python plotting routines also requires that you install h5py so that Python can read the HDF5 files.

If you run the forward modeling code in order to generate brighter-fatter plots as described in the bfrun1 example below, you will also need to build the forward.so Python extension. Instructions for this are in the forward_model_varying_i directory.

3 Changes in Most Recent Version

This 'hole18' branch is a minor revision to the 'hole17' branch revision. It is still under development and may contain bugs. However, the 'hole17' code was used to create all of the plots in the document docs/PACCD_Paper_2Feb17.pdf, so it is reasonably mature. The 'hole18' branch contains minor changes to allow simulation of the chip edge, notably to allow the parameter QFh to be different in different locations.

4 Examples

There are a total of seven examples included with the code. Each example is in a separate directory in the data directory, and has a configuration file of the form *.cfg. The parameters in the *.cfg files are commented to explain(hopefully) the purpose of each parameter. Python plotting routines are included with instructions below on how to run the plotting routines and the expected output. The plot outputs are placed in the data/*run*/plots files, so you can see the expected plots without having to run the code. If you edit the .cfg files, it is likely that you will need to customize the Python plotting routines as well.

- Example 0: data/run0/bf.cfg
  1. Purpose: This is a 'quick and dirty' file that generates a 9x9 grid of pixels with the center pixel having 80,000 electrons, at lower resolution. It should run in 10-15 minutes.
- Example 1: data/run1/bf.cfg
  1. Purpose: This is the same as run0, but at higher resolution. It will take several hours to run.
- Example 2: data/run2/bf.cfg
  1. Purpose: Similar to run1, but a 13x13 pixel array, and calculates the pixel shapes, which is slow. The file data/run2/BF_0_Vertices.dat is the most accurate simulation I have of the distorted pixel shapes, and produced the plot data/run2/plots/Area_Corr_19May17_0.pdf, which is the best fit I have to the correlation measurements.

- Example 3: data/run3/trans.cfg
    1. Purpose: This is a simulation of the output transistor, and was used to simulate the threshold volatge of this device for comparison to measurements.
- Example 4: data/run4/io.cfg
    1. Purpose: This is a simulation of the entire IO region, including the output transistor, the reset gate, and the end of the serial chain. It is still being developed, but seems to give sensible results.
- Example 5: data/run4/sat.cfg
    1. Purpose: This is what was used to predict the onset of blooming in the array as a function of the parallel gate voltages.
- Example 6: data/edgerun1/edge.cfg
    1. Purpose: To simulate astrometric shift at the edge of the pixel array.
    2. Syntax: src/Poisson data/edgerun1/edge.cfg
    3. Expected run time: $\approx$ 90minutes.
    4. Plot Syntax: python Poisson_Plots_Edge.py data/edgerun1/edge.cfg 0
    5. Expected plot run time: $<$ 1minute.
    6. Plot output: Assumed boundary potentials and charge distribution as well as several views of the potential solution.
    7. Plot Syntax: python Pixel_Shift_29Aug17.py.py data/edgerun1/edge.cfg 0
    8. Expected plot run time: $\approx$ 1minute.
    9. Plot output: Pixel shift at array edge, electron paths at array edge, and simulated roll-off of spot location.