Poisson Solver Code

Craig Lage

October 1, 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 Cori.

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 bfrun 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 'hole20' branch is a fairly major revision, with a number of new features. In addition, it has been tested much more extensively, and many of the examples listed in the next section have been verifed with actual data taken on the STA3800 detector. The Poisson_30Sep17.pdf presentation in the docs directory has a presentation-style summary of these results. The following is a list of the changes:

- Three different methods for adding electrons to the CCD pixels have now been included, and these are selected with the ElectronMethod parameter. A description and comparison of these three methods is given in slides 64-68 of the Poisson_30Sep17.pdf presentation in the docs directory.
- A ChannelStopSideDiff parameter has been added to allow independent control of side diffusion of the channel stops. The default is that this is equal to the FieldOxideTaper parameter, which is how it ran in the past.
- Several parameters have been added to allow tree rings to be added to the simulation. The treering example below will make their usage clear.
- An option has been added to simulate fringes projected onto the CCD in addition to the spots that were used before. This is selected with PixelBoundaryTestType = 3, as in the fringerun example below.
- The option to save all of the coarser multi-grids used in the multi-grid solver has been added with the SaveMultiGrids parameter. This is useful when testing convergence. The smallpixel and smallcap examples below illustrate the usage.
- The plotting routines have been modified fairly extensively, and all placed in a pysrc subdirectory. Most of the subroutines used by these plotting routines have been collected in the pysrc/pysubs.py file. This has eliminated a lot of duplication that was present in the past.
- A GateGap parameter has been added to study the impact of gaps between the gate electrodes. This should be considered experimental at this time, and is not included in any of the examples below.

4 Examples

There are a total of thirteen 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 to generate plots for each example. The plot outputs are placed in the data/*/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 may need to customize the Python plotting routines as well.

A new feature that has been added is the Run_Tests.py script which is included with this release. This script will run all of the examples and generate all of the plots which are included in the data/*/plots/ directories. Less detail has been included on the syntax for running each example and the associated plots, since you should be able to see this from the Run_Tests.py script.

- Example 1: data/smallpixel/smallpixel.cfg
    1. Purpose: This is a single pixel, and typically runs in a few seconds. It is useful for set-up and debug.
- Example 2: data/pixel0/pixel.cfg
    1. Purpose: This is a 9x9 pixel array with the central pixel having 80,000 electrons, It is a low resolution, and typically runs in < 1 minute. Again, it is useful for set-up and debug.
- Examples 3, 4, 5: data/pixel$< 1 >< 2 >< 3 >$/pixel.cfg
    1. Purpose: This is a 9x9 pixel array with the central pixel having 80,000 electrons, It is at ScaleFactor=2 resolution, which typically is sufficient accuracy. These three runs test the three different Electron-Method options, and plot out the distorted pixel shapes. pixel1 and pixel2 take about 45 minutes each, while pixel3 takes several hours, since it is building up the electron cloud in ≈ 100 steps.
- Example 6, 7: data/smallcap$< g >< f >$/smallcap.cfg
    1. Purpose: These test convergence on gate and field oxide capacitors.
- Example 8: data/bfrun/bf.cfg
    1. Purpose: This example builds a spot to test the brighter-fatter effect. The Run_Tests.py script builds a number of spots with dithered starting locations, which are averaged to reduce Poisson noise.
- Example 9: data/edgerun/edge.cfg
    1. Purpose: This example tests the astrometric roll-off at the edge of the pixel array.
- Example 10: data/satrun/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. The Run_Tests.py builds a total of 24 different directories and runs a 4x6 matrix of parallel and serial voltages.
- Example 11: data/transrun/trans.cfg
    1. Purpose: This is used to simulate the turn-on characteristics of the output transistor. The Run_Tests.py script builds a total of 13 directories with different gate voltages.
- Example 12: data/iorun/io.cfg
    1. Purpose: This is a simulation of the end of the serial chain as it feeds into the I/O transistor on the STA3800 device.

- Example 13: data/treering/treering.cfg

    1. Purpose: This is an example of adding tree rings to the simulation by modulating the background doping. The modulation is assumed to vary only in the X and Y dimensions and be constant in Z.