Poisson_CCD: A dedicated simulator for modeling CCDs

Craig Lage, Andrew Bradshaw, J. Anthony Tyson
Department of Physics
University of California - Davis
cslage@ucdavis.edu

Abstract

Keywords: LSST, modeling, camera, CCD, simulation, diffusion, image processing.

A dedicated simulator, Poisson_CCD, has been constructed which models astronomical CCDs by solving Poisson's equation numerically and simulating charge transport within the CCD. The potentials and free carrier densities within the CCD are self-consistently solved for, giving realistic results for the charge distribution within the CCD storage wells. The simulator has been used to model the CCDs which are being used to construct the LSST digital camera. The simulator output has been validated by comparing its predictions with several different types of CCD measurements, including astrometric shifts, brighter-fatter induced pixel-pixel covariances, saturation effects, and diffusion spreading. The code is open source and freely available.

1 Introduction

Charge Coupled Devices (CCDs) have been the workhorse devices for astronomical imaging for some time. George Smith's Nobel lecture at [1] gives an excellent summary of the early history. While other detectors are making inroads, CCDs are still the dominant imaging device in astronomical applications. In recent years thick, fully depleted CCDs with their wide spectral response have been applied to spectroscopic applications as well as imaging. Although these devices have high quantum efficiency, relatively good linearity, and acceptable dynamic range, they have a number of problematic effects that can impact the precision and accuracy of astronomical data. It is important that these effects are well understood so that they can be removed during image processing. To help understand these effects, we have built a dedicated simulator, Poisson_CCD, which solves the electrostatics in the bulk silicon of the CCD, and propagates incoming charges down to the collecting wells where they are collected and stored. The simulator has proven very useful for understanding a number of CCD effects, which will be described in this paper.

Of course, detailed semiconductor modeling codes already exist, are commercially available, and have been validated against silicon results. What is the purpose of developing yet another simulator? The answer is severalfold. First, commercial semiconductor codes typically use proprietary source codes and are quite expensive, while the code described here is open source and freely available. Second, using a commercial semiconductor device simulator requires spending quite a bit of time learning to use the code and set up the initial conditions. The code described here sets up the initial conditions for a typical CCD with a few simple configuration parameters. Also, it is hoped that the code is simple enough that it can be mastered by people who are not semiconductor experts. The target user group is people in the astronomy field who want to answer questions about CCDs without investing a great deal of time learning the details of semiconductor physics.

The code was developed as part of the development effort of the LSST. This instrument is an innovative, large, fast survey facility currently under construction at Cerro Pachon in Chile [2]. The digital camera for the LSST, also currently under construction, will consist of approximately 3.2 gigapixels and will be the largest digital camera

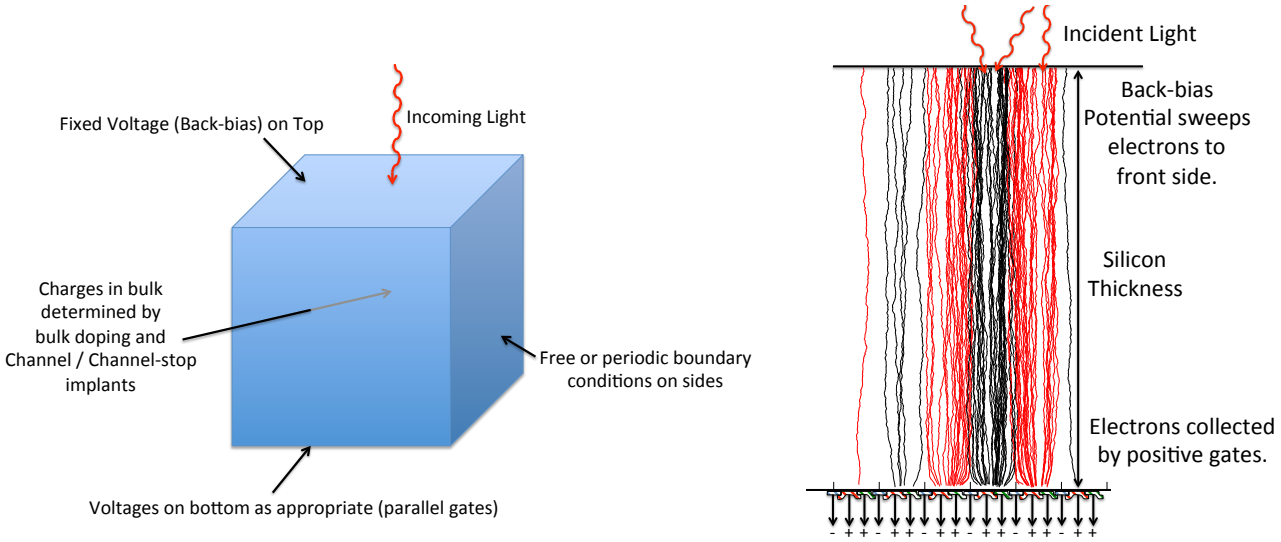
ever constructed. The camera uses fully-depleted silicon CCDs which are back illuminated and 100 microns thick in order to optimize quantum efficiency in the near infrared. The imaging area consists of 189 CCDs, with each CCD containing 16 imaging regions laid out in an 8x2 array. Each imaging region has a pixel array with approximately 500x2000 10 micron square pixels, giving 16 Megapixels total. Each imaging region also has its own independent amplifier ([3], [4]). The LSST focal plane contains CCDs from two different vendors, the ITL STA3800C from the University of Arizona Imaging Technology Laboratory [5], and the E2V CCD250 from Teledyne E2V [6]. However, although this code was developed and tested against these two CCDs from the LSST project, it has already found more general use on other CCDs ([7]), and the hope is that this will continue.

This paper is divided into several sections. In the first section, we give an overview of the simulator, describing the basic structure of the simulation volume, how we solve the semiconductor equations, and how we treat incoming photons. We also show a number of examples of the outputs available from the simulator. In the second section, we review a number of the validation tests that were performed to validate the simulation results against measured data of different kinds, and finally we conclude.

2 Overview

The simulator performs two basic tasks, as shown in Figure 1. First, given the charges in the silicon bulk and the boundary conditions determined by potentials applied to the silicon surface, the simulator solves Poisson's equation numerically to determine the potentials and electric fields in the silicon bulk. The solution to Poisson's equation is determined using the technique of successive over-relaxation(SOR), and using multi-grid methods to speed convergence. In regions where there are mobile carriers (holes and electrons) quasi_Fermi level methods are used to simultaneously solve for the potentials and free carrier densities in the device. The electric fields are determined by numerically differentiating the electrostatic potential. In general, the simulator only solves for the potentials and free-carrier densities in equilibrium, and is not intended to give transient solutions. However, it is possible to repeatedly solve the equations with slight changes in initial conditions in order to give transient results. This technique has been used to generate movies of the CCD charge transport, as discussed in Section 3.6.

After solving for the device potentials, the second major task of the simulator comes into play. As incoming photons enter the CCD, they generate hole-electron pairs. The electric field in the CCD separates these charges, and the electrons propagate down to the collecting wells where they are collected, stored, and later counted. The simulator models this carrier transport in a physically realistic way, in order to determine in which pixel a generated carrier ends up. This is very useful for modeling pixel distortions that results from electric fields in the device, either built in electric fields, such as those due to "tree rings"[8], or electric fields due to collected charges, such as those that lead to the brighter-fatter (BF) effect ([9], [10], [11], [12], [13]). Note that in an astronomical CCD, the time between incoming charges (on the order of msec) is typically much longer than the time required for a charge to propagate down to the bottom (on the order of nsec). Also, a single charge has little impact on the existing potentials and fields. So it is an excellent approximation to assume that a single charge propagates in a frozen electric field. The simulator is designed so that if multiple charges are being added, the user can choose how often to re-solve Poisson's equation. A comment here about the nature of the charges is in order. The simulator is designed for N-type CCDs, where the collected charges are electrons, and for the rest of the paper we will make that assumption. There is no physical reason why it will not work for P-type CCDs, but it is not currently designed to support them. Most of the work in adapting it for P-type CCDs would be nomenclature related, basically interchanging the names of holes and electrons. This could be done if there is demand for it.



(a) Silicon volume (b) Charge tracking

Figure 1: Major tasks of the Poisson_CCD simulator. The left hand figure shows the silicon volume. Given charges in the silicon bulk and voltages applied to the silicon boundaries, one solves for the electric fields and free carrier densities within the silicon. The right hand figure shows incoming charges, created by incident photons, which are propagated down through the silicon bulk to determine the number of charges in each pixel.

2.1 Basic structure of the simulations

The simulator is written in C++, and is controlled by a text-based configuration file, which contains all of the information about the silicon volume, pixel sizes, number of pixels, any non-pixel regions, etc. The configuration file also defines the problem being solved. By convention the configuration file has a .cfg extension, but this is not necessary. Appendix A lists the configuration parameters. As the simulation progresses, it writes out a number of files. Large files containing information like potentials, charge densities, electric fields at each grid point are written as high-density HDF5 files, having file extension .hdf5. Several smaller text files, with a .dat extension, are written which contain information on the grids or the number of electrons in each pixel. After the simulation has completed, a series of easily modifiable Python scripts are used to plot out results as desired.

The simulation volume is set up on a fixed three dimensional rectangular grid, which does not change once the simulation has started. One begins by deciding the number of grid cells in each dimension. Because multi-grid methods are used (see Section 2.4), the number of grid cells in each dimension must be a multiple of 32. Note that as a convention, we refer to the side of the CCD where the circuitry is patterned as the bottom, and the side where the incident light comes in as the top. For the LSST CCDs, which are 100 microns thick and have pixels 10 microns square, a typical resolution is to have 32 simulation grid cells per pixel, so that each grid cell is 0.31 microns. Initially, the grid was defined to be completely uniform in all three dimensions. However, for thick CCDs like those used in the LSST, the potentials and fields change rapidly in the region near the bottom, and only very slowly near the top. When the simulation had enough resolution to be accurate in the rapidly changing region at the bottom, most grid cells were wasted near the top. Of course, adaptive grid methods solve this problem, but also make the code much more complex, which defeated the purpose of having a relatively simple simulator. The solution chosen in this work was to use a non-linear grid in the Z-dimension only. This has proven to give a high resolution where needed, without adding significant complexity to the simulation code. Figure 2 shows the scheme. This introduces some additional partial derivatives, as discussed more in section 2.3, but these values can be pre-calculated and this is much simpler than an adaptive grid scheme.

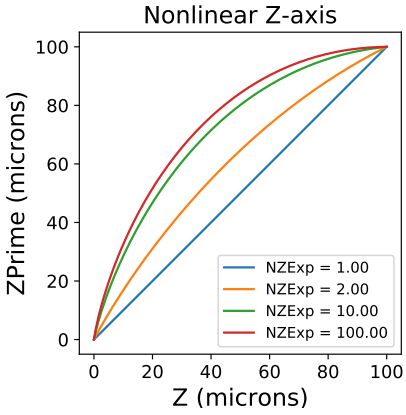


Figure 2: Non-linear Z-axis scheme. The parameter NZExp allows one to have smaller Z-axis grid cells near the bottom of the simuation volume, where the fields and charges are changing more rapidly. A value NZExp=1 is a linear grid. The recommended value is to Use NZExp=10.0, which increases the resolution at z=0 by a factor of 10, and decreases the resolution at the top of the CCD by a factor of 10.

2.2 Setting up the initial conditions

Because the simulator is not a general purpose semiconductor solver, and is intended to model devices with a given structure, certain assumptions are made about the structure of the CCD which simplifies building the device structure. It is assumed that the CCD is a slab of silicon with a given thickness given by the parameter "SensorThickness". It is assumed that the top surface of the CCD is at a fixed voltage given by the parameter "Vbb". The bottom surface of the CCD has voltages specified by the various gate potentials. The doping deep in the silicon is asumed constant with a value given by "BackgroundDoping", although a periodic variation in this doping can be introduced using the "TreeRing" parameters. The doping level is assumed to be modified by the introduction of implants from the bottom side. There are several options for these doping profiles, including a square profile of a given depth or a sum of N Gaussian profiles. Use of 1 or 2 Gaussian profiles has been found to accurately reproduce the measured doping profiles on commercial CCDs. For more details on this, see [14].

2.2.1 Pixel arrays

Setting up the initial conditions in the periodic pixel array is straightforward, and is specified by a relatively small number of parameters which describe the gate voltages and doping levels. Rather than go through these in detail, the reader is referred to Appendix A or the "pixel-itl" and "pixel-e2v" examples at [15].

2.2.2 Fixed regions

Setting up the initial conditions in non-periodic regions ouside the pixel array is straightforward, but more laborious than setting up the pixel arrays. The extents, dopings, applied voltages, and quasi-Fermi levels need to be specified for each region. At present only rectangular regions are supported. Also, it is assumed that the same doping profiles which are used in the pixel array are used in the surrounding circuitry, so the only options for doping profiles are the channel doping, the channel stop doping, and no doping. Examples of simulations setting up non-periodic regions are the "edge.cfg", "trans.cfg", and "io.cfg" files at [15], and the results of these simulations are detailed in Section 3.

2.3 Solving for the potentials, fields, and free carrier densities

In this section we give a brief description of the methods that are used to solve Poisson's equation on the grid. We are trying to solve:

$$\nabla^2\varphi = \rho \tag{1}$$

Each of the partial derivatives can be discretized as follows:

$$\frac{\partial^2\varphi_{\mathrm{i,j,k}}}{\partial \mathrm{x}^2} = \frac{(\varphi_{\mathrm{i+1,j,k}} - \varphi_{\mathrm{i,j,k}}) - (\varphi_{\mathrm{i,j,k}} - \varphi_{\mathrm{i-1,j,k}})}{\mathrm{h}^2} \tag{2}$$

Giving:

$$(\varphi_{\mathrm{i+1,j,k}} + \varphi_{\mathrm{i-1,j,k}} + \varphi_{\mathrm{i,j+1,k}} + \varphi_{\mathrm{i,j-1,k}} + \varphi_{\mathrm{i,j,k+1}} + \varphi_{\mathrm{i,j,k-1}} - 6 * \varphi_{\mathrm{i,j,k}}) = \mathrm{h}^2 * \rho_{\mathrm{i,j,k}} \tag{3}$$

This can be turned into an iterative equation, and basically one just iterates until convergence:

$$\varphi^{(\mathrm{n+1})}_{\mathrm{i,j,k}} = \frac{1}{6} * (\varphi^{(\mathrm{n})}_{\mathrm{i+1,j,k}} + \varphi^{(\mathrm{n})}_{\mathrm{i-1,j,k}} + \varphi^{(\mathrm{n})}_{\mathrm{i,j+1,k}} + \varphi^{(\mathrm{n})}_{\mathrm{i,j-1,k}} + \varphi^{(\mathrm{n})}_{\mathrm{i,j,k+1}} + \varphi^{(\mathrm{n})}_{\mathrm{i,j,k-1}} - \mathrm{h}^2 * \rho_{\mathrm{i,j,k}}) \tag{4}$$

Which we write in shorthand as follows:

$$\varphi^{(\mathrm{n+1})} = \frac{1}{6} * (\varphi^{(\mathrm{n})}_{\mathrm{pm}} - \mathrm{h}^2 * (\rho_{\mathrm{f}} + \rho_{\mathrm{m}})) \tag{5}$$

where we define:

$$\varphi^{(\mathrm{n})}_{\mathrm{pm}} = \varphi^{(\mathrm{n})}_{\mathrm{i+1,j,k}} + \varphi^{(\mathrm{n})}_{\mathrm{i-1,j,k}} + \varphi^{(\mathrm{n})}_{\mathrm{i,j+1,k}} + \varphi^{(\mathrm{n})}_{\mathrm{i,j-1,k}} + \varphi^{(\mathrm{n})}_{\mathrm{i,j,k+1}} + \varphi^{(\mathrm{n})}_{\mathrm{i,j,k-1}} \tag{6}$$

and we have split ρ into a fixed charge density ρ_{f} and a mobile charge density ρ_{m}. However, ρ_{m} is a highly nonlinear function of the potential φ, as described below. In quasi-equilibrium, the drift and diffusion currents are equal, giving a net current of zero, so we can write (see Sze [16], for example):

$$\mathrm{J_E} = \mathrm{q_e}\mu_{\mathrm{n}}\mathrm{n}\frac{\mathrm{d}\varphi}{\mathrm{dx}} = -\mathrm{J_D} = -\mathrm{q_e}\mathrm{D_n}\frac{\mathrm{dn}}{\mathrm{dx}} \tag{7}$$

$$\mu_{\mathrm{n}}\mathrm{d}\varphi = -\mathrm{D_n}\frac{\mathrm{dn}}{\mathrm{n}} \tag{8}$$

and we have:

$$\frac{\mu_{\mathrm{n}}}{\mathrm{D_n}} = \frac{\mathrm{q_e}}{\mathrm{kT}} \text{ (Einstein Relation)} \tag{9}$$

so:

$$\frac{\mathrm{q_e}\varphi}{\mathrm{kT}} = \log(\mathrm{n}) + \mathrm{C} \tag{10}$$

We take the constant of integration into the exponential and define the quasi-Fermi level φ_{F} in terms of the intrinsic carrier density $\mathrm{n_i}$, giving:

$$\mathrm{n} = \mathrm{n_i}\exp(\frac{\mathrm{q_e}(\varphi - \varphi_{\mathrm{F}})}{\mathrm{kT}}) \tag{11}$$

So we need to solve the following equation for φ, where φ_{F} is a constant:

$$\nabla^2\varphi = \frac{1}{\epsilon_{\mathrm{Si}}}(\rho_{\mathrm{f}} + \mathrm{q_e}\mathrm{n_i}\exp(\frac{\mathrm{q_e}(\varphi - \varphi_{\mathrm{F}})}{\mathrm{kT}})) \tag{12}$$

which, when discretized is:

$$\varphi^{(\mathrm{n+1})} = \frac{1}{6} * (\varphi^{(\mathrm{n})}_{\mathrm{pm}} - \mathrm{h}^2 * \rho_{\mathrm{f}} - \mathrm{h}^2\mathrm{K}\exp(\frac{\mathrm{q_e}(\varphi^{(\mathrm{n+1})} - \varphi_{\mathrm{F}})}{\mathrm{kT}}))) \tag{13}$$

Because of the strong non-linearity, simply iterating is numerically unstable. The method that works, originally pioneered by Rafferty, et.al. [17] is to take this last equation as a non-linear equation for $\varphi^{(\mathrm{n+1})}$ in terms of

φ^{n} and run a Newton's method "inner loop" to find $\varphi^{(\mathrm{n}+1)}$ at each grid point. Then we iterate to convergence as before. This allows us to simultaneously solve for the potential and the carrier density. Of course, we just described the electron density here, but there is a similar equation for holes, but with opposite signs. Figure 3 shows an example of varying φ_{F} on the solution. Note that the quasi-Fermi level is constant in each region containing mobile carriers. In the CCD, each collecting well contains a different number of mobile carriers, so φ_{F} is constant in each well, but is different from well to well. However, when simulating the device, instead of knowing the value of φ_{F}, we typically know the number of electrons in each well. So how do we translate from the known number of electrons to the unknown value of φ_{F}? The code provides two methods, selected by the value of the parameter "ElectronMethod". With this parameter set to 1, a test simulation is run where the parameter φ_{F} (called QFe in the code) is varied through a range and then the code interpolates to determine the value of QFe which gives the appropriate number of electrons. In practice one can get close to the desired number of electrons, but the non-linearity causes variations from the desired number, so a second method was developed. When "ElectronMethod" has a value of 2, what is done is to place the correct number of electrons in the well, uniformly distributed in the center of the well. The code then moves the electrons around until the value of QFe is constant in the well. This allows one to get the correct number of electrons in the well without needing to know the value of the quasi-Fermi level.

There is one more complication. As discussed in Section 2.1, a non-linear Z-axis is used to concentrate grid cells in the bottom region where the potentials and charge demsities are changing much more rapidly. In principle, any smooth funcation can be used for the Z-axis mapping. What was chosen here is an easily differentiable polynomial function of the following form, where z is the linear z coordinate, and zp is the non-linear coordinate, which is the actual value used in solving and plotting.

$$\mathrm{zp} = -\mathrm{T_{Si}} * (\mathrm{NZExp} - 1.0) * (\mathrm{z}/\mathrm{T_{Si}})^{(\mathrm{NZExp}+1.0)/\mathrm{NZExp}} + \mathrm{NZExp} * \mathrm{z}; \tag{14}$$

The non-linear z-axis modifies Poisson's equation from:

$$\nabla^2\varphi = \frac{\partial^2\varphi}{\partial \mathrm{x}^2} + \frac{\partial^2\varphi}{\partial \mathrm{y}^2} + \frac{\partial^2\varphi}{\partial \mathrm{z}^2} \tag{15}$$

to:

$$\nabla^2\varphi = \frac{\partial^2\varphi}{\partial \mathrm{x}^2} + \frac{\partial^2\varphi}{\partial \mathrm{y}^2} + \frac{\partial^2\varphi}{\partial \mathrm{z}^2}(\frac{\partial \mathrm{z}'}{\partial \mathrm{z}})^2 + \frac{\partial\varphi}{\partial \mathrm{z}'}\frac{\partial^2 \mathrm{z}'}{\partial \mathrm{z}^2} \tag{16}$$

In practice, the added partial derivatives can be pre-computed, so this is simply added into the discretized equations and has only a minor impact on the speed of iteration.

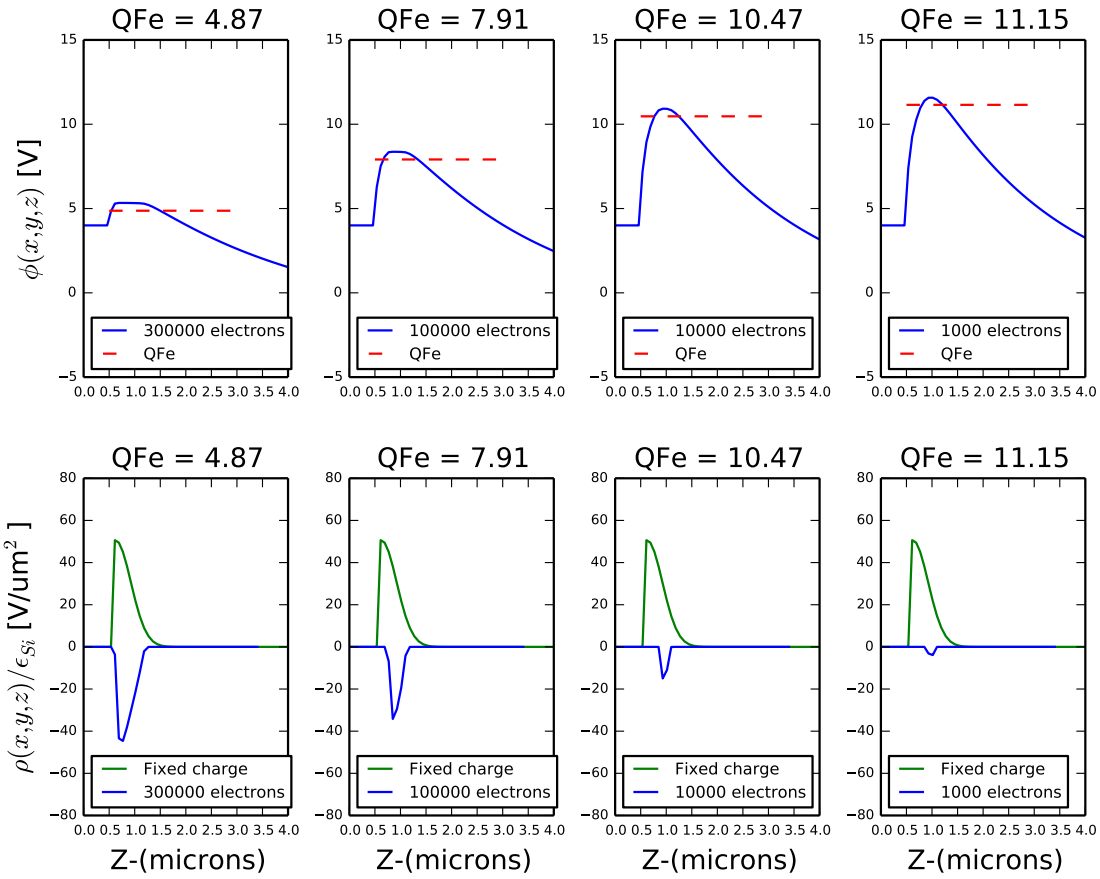
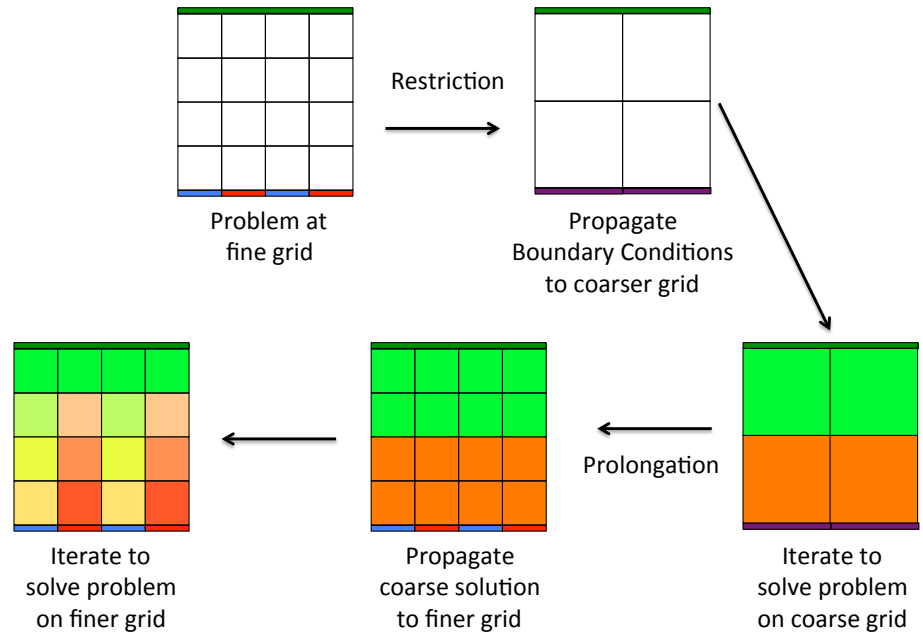


Figure 3: Impact of varying φ_F (called QFe in the code) on the potential and electron density.

2.4 Multi-grid methods

It is well known that multi-grid methods speed convergence of solutions to Poisson's equation by getting correct solutions to the long-wavelength modes at a coarser grid where convergence is much more rapid. There is a wealth of literature on the subject, and Briggs [18] or Press [19] give excellent summaries. The basic idea is shown in Figure 4. In practice in this code, we have adopted a simpler method. Rather than use "Restriction" to propagate the boundary conditions down to the coarser grid, we simply set up the boundary conditions on all of the sub-grids at the outset of the problem. In addition, we have found that there is little value in running coarser grids than 40^3, because at this resolution the problem converges very rapidly. So for a typical problem which has perhaps 320^3 grid cells, we define the finest grid and three subgrids, with the coarsest grid having 40^3 grid cells. We then set up the boundary conditions on all of the subgrids, iterate the coarsest grid, "Prolongate" the solution up to the next grid, and continue until we have reached the finest grid.



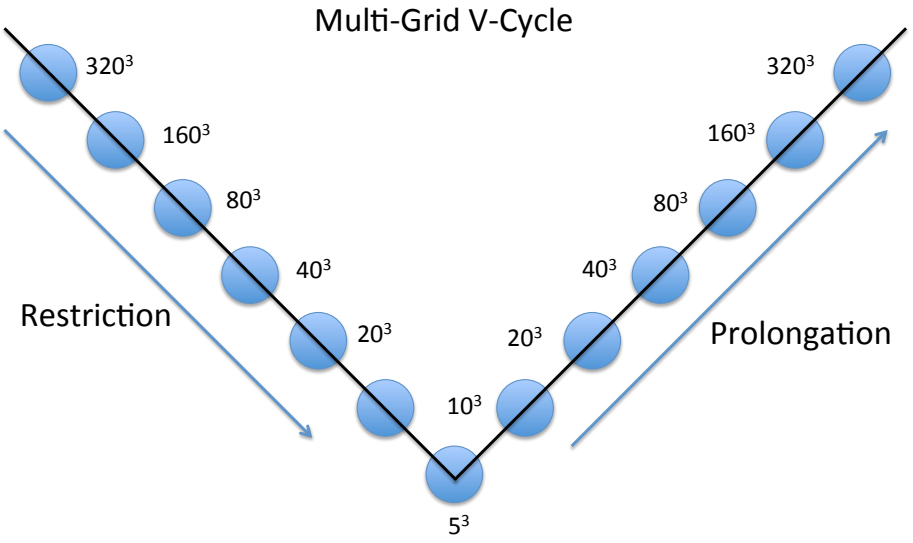


Figure 4: Basic idea of multi-grid methods. Long wavelength modes are solved on a coarser grid, which is then propagated to a finer grid ("Prolongation"), where more iterations are performed to find the fine details of the solution.

At this point it is appropriate to discuss the problem of convergence. Convergence of the SOR algorithm is notoriously slow. Multi-grid methods help a great deal, however, care must still be taken to ensure that the solution has converged adequately for your problem. The parameter "ncycle" controls the number of iterations taken at the finest grid. Each coarser grid increases the number of iterations by a factor of 4. So for example, a

typical problem like one of the "pixel" examples, which has ncycle=128, the coarsest grid has 1/8 the resolution, and will run $128 \times 4^3 = 8192$ SOR cycles at the coarsest grid. Figures 5 and 6 show the convergence of a typical problem. For most problems, a value of ncycle=64 is adequate. The most sensitive problems have proven to be the pixel distortion simulations like those in Section 3.1. Since we are dealing with very small deviations in the pixel shapes due to the BF effect, it is important to make sure the results have converged. Note that if the "VerboseLevel" parameter is 2 or larger, the program will print out the SOR error at each multi-grid. This is the largest change to the potential (in volts) that was made at the last SOR iteration. One should continue the iterations until this error result is millivolts or less.

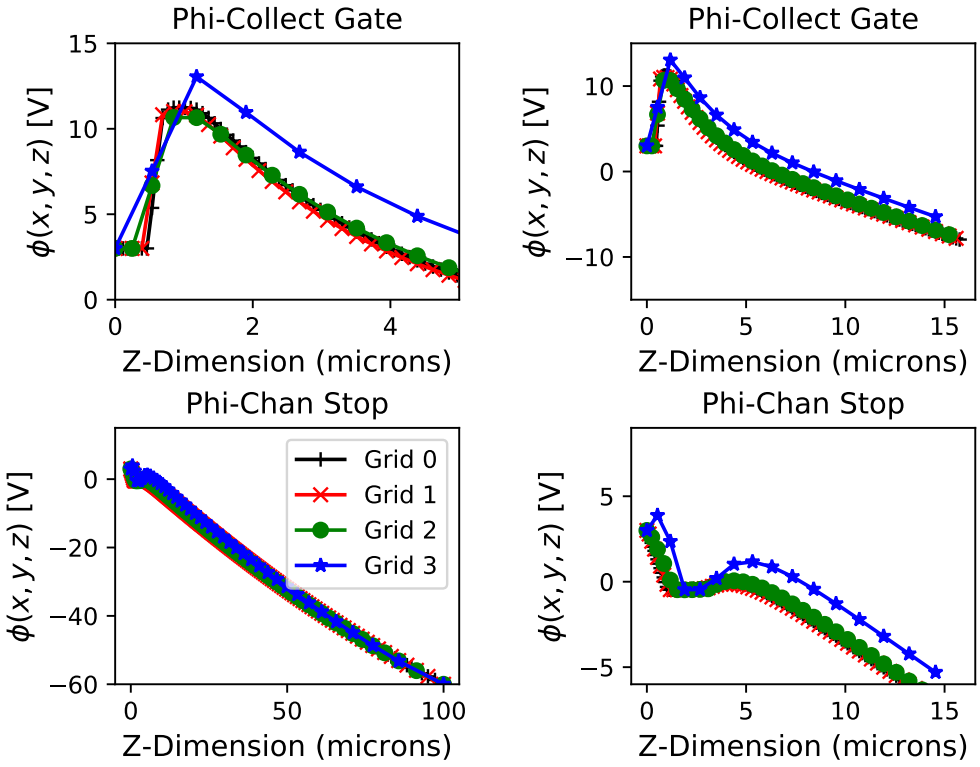


Figure 5: Convergence of the multi-grid subgrids. The parameter "ncycle" has a value of 64 in these plots. Each subgrid is a factor of two coarser than the preceding grid.

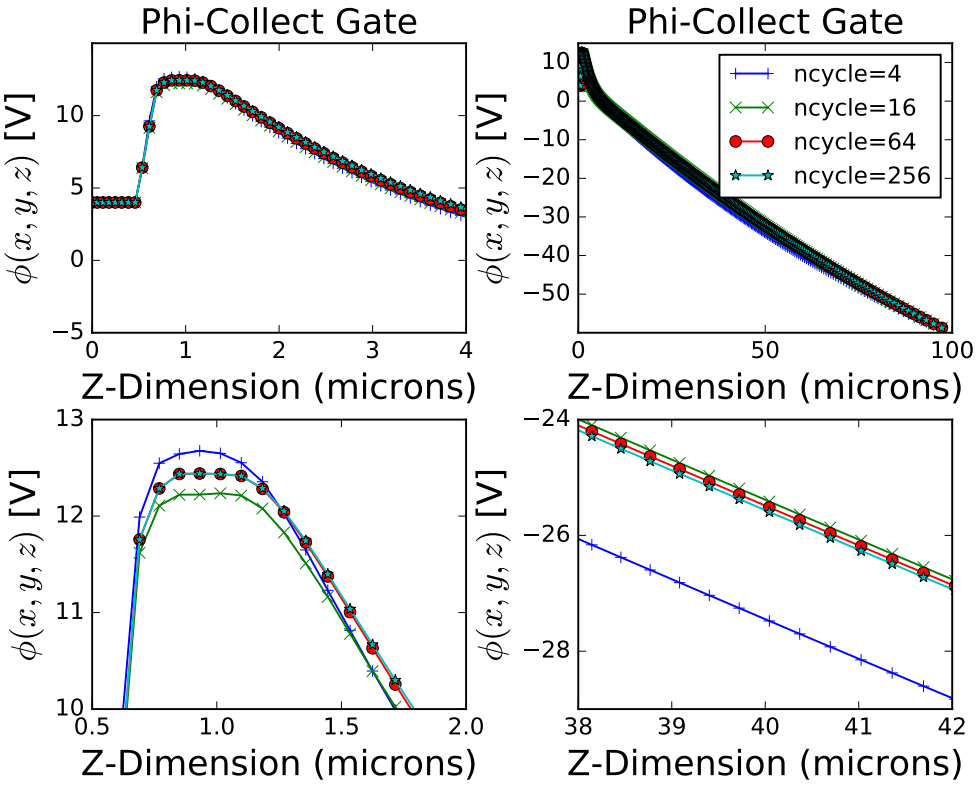
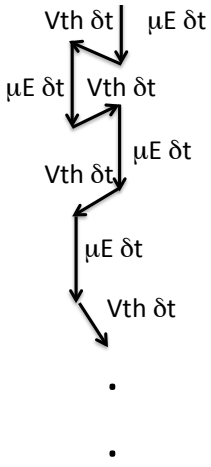


Figure 6: Convergence of the highest resolution grid as a function of the parameter "ncycle". For most uses a value of 64 is adequate.

2.5 Modeling carrier transport

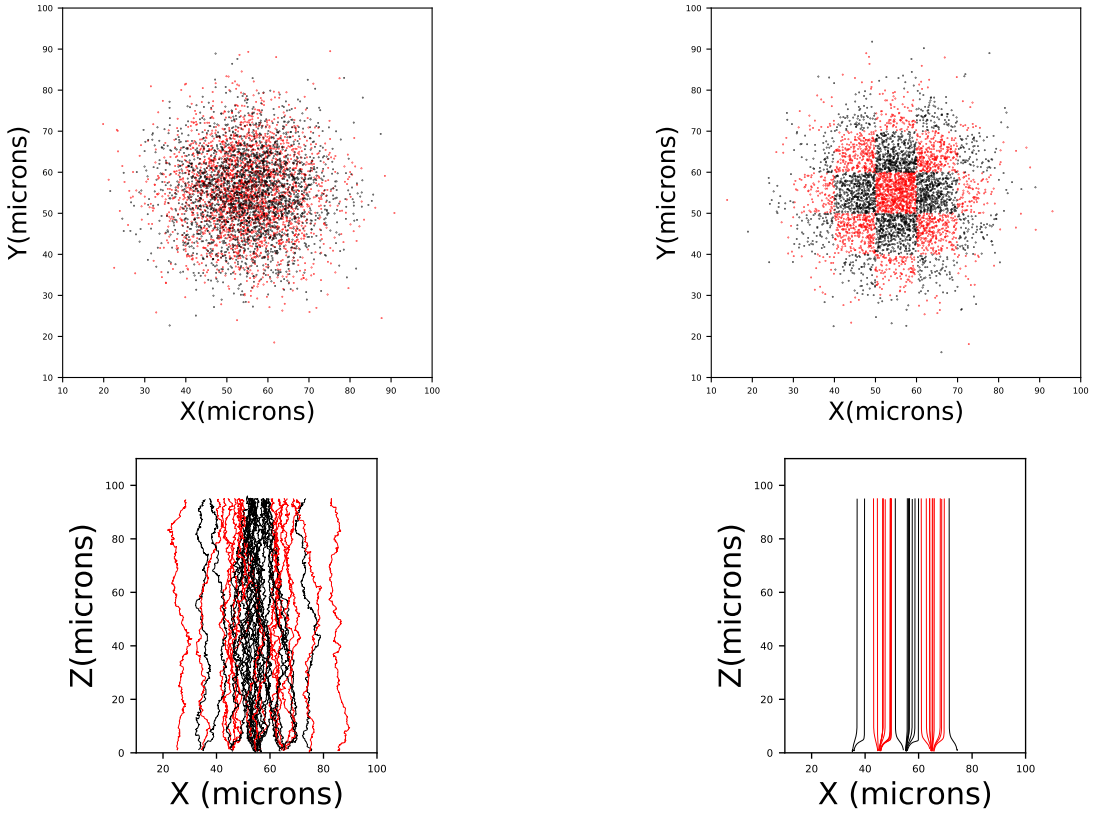
The basic scheme for modeling carrier transport is shown in Figure 7. Electrons are assumed to have lattice collisions on a time scale τ, which is on the order of picoseconds. At each collision, the electron is assumed to pick up a thermal velocity V_{th} which is in a random direction. In addition to this thermal velocity, it also has a drift velocity given by $V_{drift} = \mu E$. These two velocities are added vectorially and it travels in this direction at this velocity for a time δt until the next collision, and this continues until the electron reaches the bottom. The electron path is logged in the *_Pts.dat file. If the parameter "LogPixelPaths" is zero, only the initial and final positions are logged. If this parameter is one, the entire path is logged. The thermal velocity has a multiplier ("DiffMultiplier") which can be used to tune the amount of diffusion. If this is set to zero, diffusion is turned off, with the impact as shown in Figure 8. A value DiffMultiplier = 2.30 has been found to accurately reproduce the amount of diffusion seen in Fe^{55} data (see Section 3.2). Since the value of m_e^* in the code is the bare electron mass, this value is equivalent to an electron effective mass of about 0.19. This is somewhat low, as Green [20] finds a value of 0.27. This value of DiffMultiplier is easily adjusted by the user, however.

The initial electron locations in X and Y can be determined in a number of ways, as determined by the "PixelBoundaryTestType" parameter. These include an equally spaced grid, a Gaussian spot, a random location within a boundary, an Fe^{55} event, or reading in a list of locations. The starting location in Z can either be specified, or calculated given a filter band.



Mobility: $\mu(E,T)$ calculated from Jacobini model [21]
$\mu \approx 1500 \frac{\mathrm{cm}^2}{\mathrm{V-sec}}$ at $\mathrm{E} = 6000 \frac{\mathrm{V}}{\mathrm{cm}}$
Collision time:
$\tau = \frac{\mathrm{m}_\mathrm{e}^*}{\mathrm{q}_\mathrm{e}} \mu$
τ typically about 0.9 ps.
δt drawn from exponential distribution with mean of τ
$\mathrm{V}_{\mathrm{th}} = \mathrm{DiffMultiplier} \sqrt{\frac{8\mathrm{kT}}{\pi \mathrm{m}_\mathrm{e}^*}}$
$\mathrm{V}_{\mathrm{drift}} = \mu \mathrm{E}$
Each thermal step in a random direction in 3 dimensions.
Typically about 1000 steps to propagate to the collecting well.

Figure 7: Diffusion model



Realistic Diffusion

Diffusion turned off

Figure 8: Impact of diffusion on electron paths of a Gaussian spot with a sigma of 1 pixel. With diffusion turned off, the electrons simply propagate down and end up in the same pixel they started. With realistic diffusion, the electrons can cross pixel boundaries.

2.6 Example outputs

Once the simulation has run, the potential, electric fields, and charge carrier densities are available throughout the simulation volume. What one chooses to visualize depends on the problem being studied. Here we have chosen three examples of the type of data which is available.

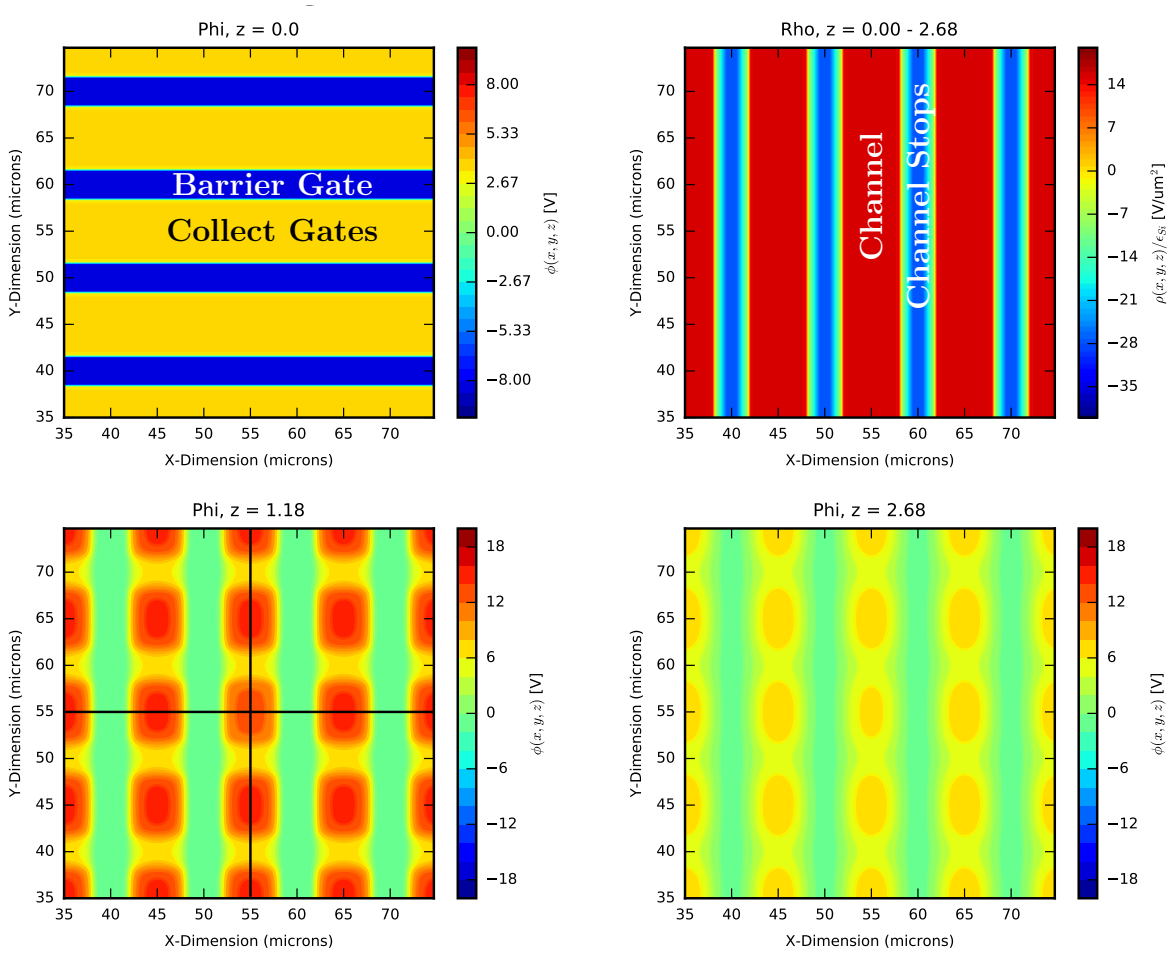


Figure 9: A summary of the region near the bottom of the ITL STA3800C CCD. The upper left shows the applied parallel gate voltages, and the upper right shows a 2D projection of the fixed charges. The lower two plots show the potential at two different z-values above the bottom of the CCD. Here the center pixel has 80,000 electrons and the surrounding pixels are empty.

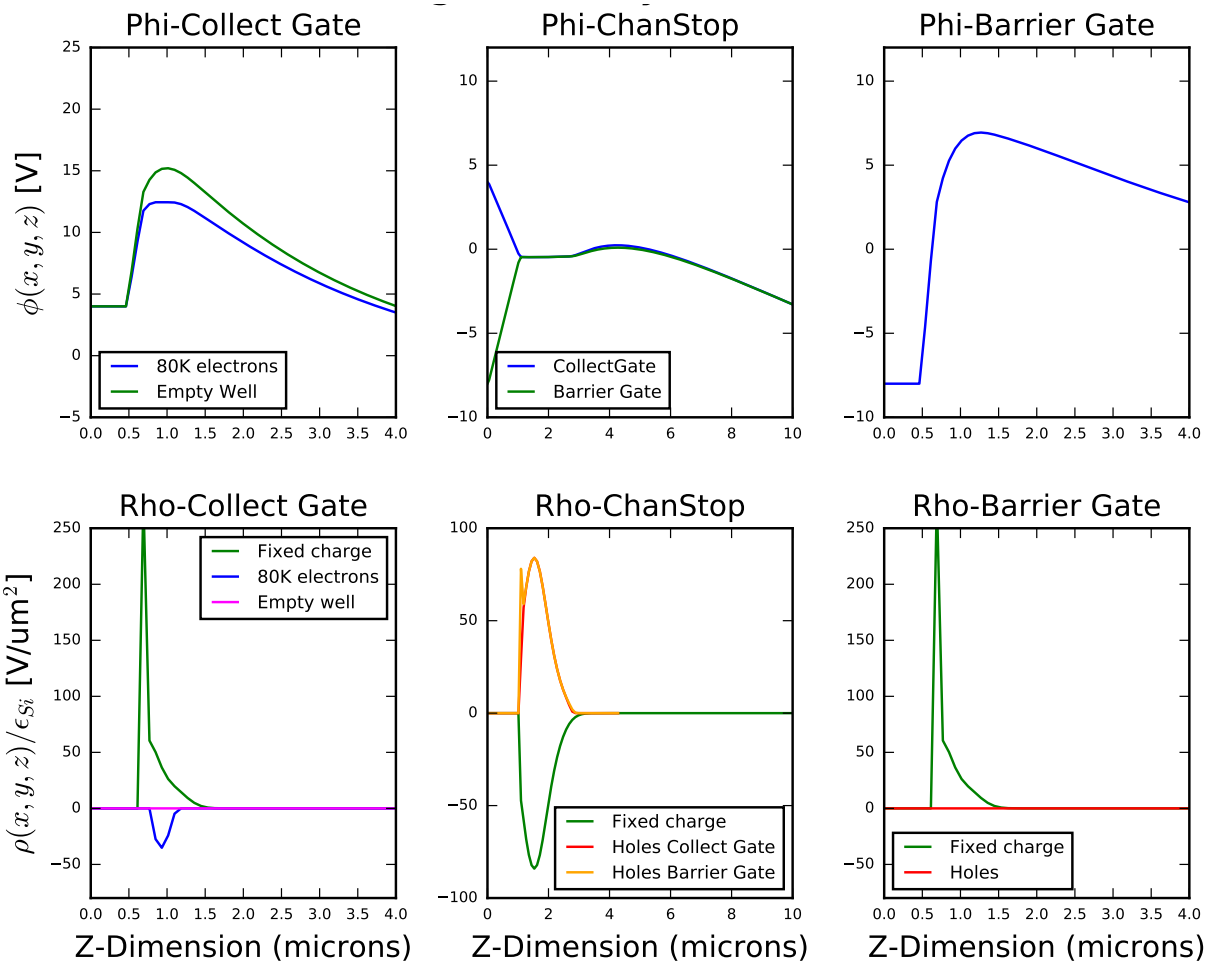


Figure 10: A set of vertical 1D profiles of potential and charge density at various locations of the ITL STA3800C CCD.

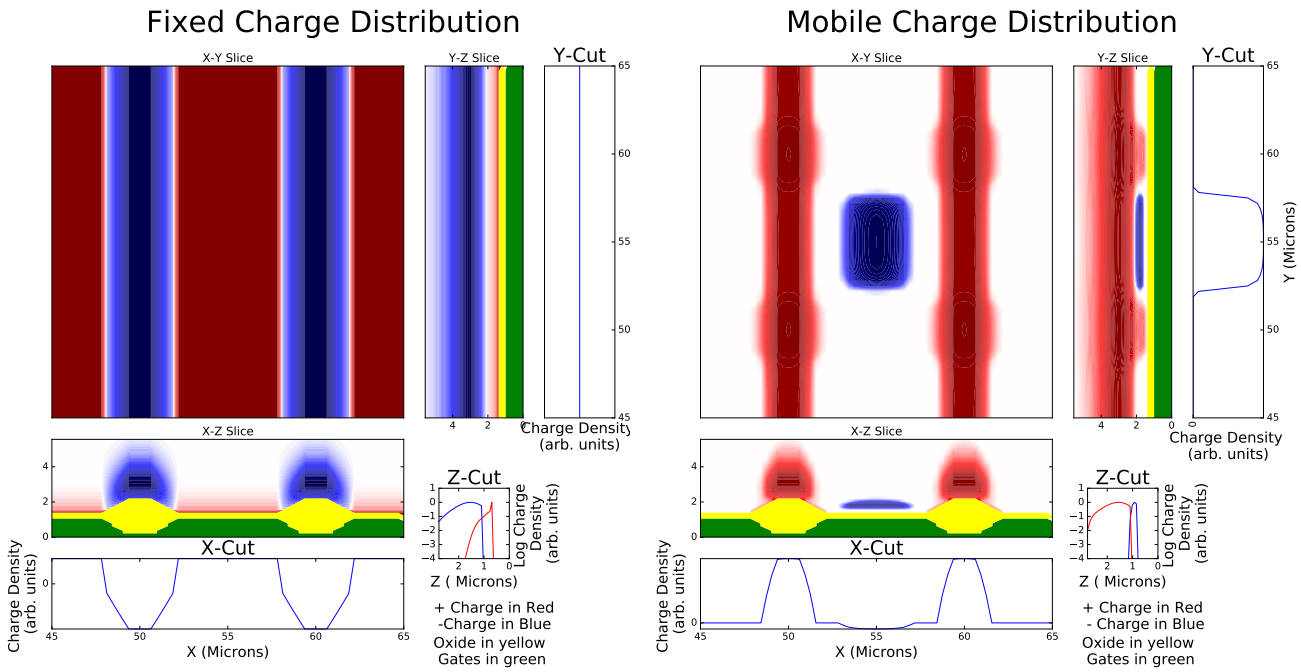


Figure 11: A set of 2D projections of the distribution of fixed and mobile charges near the bottom of the ITL STA3800C CCD.

3 Validation with measured data

In order to successfully model a CCD, it is important to have physical measurements of the CCD in order to inform the simulations. Details such as dopant densities, layer thicknesses, and physical dimensions need to be known, at least approximately. For modeling the CCDs from the two vendors which will be used in the LSST camera, we obtained detailed physical measurements, which have been described in [14]. This has helped to make the simulations described in the next few sections as physically realistic as possible. Most of the plots in this section can be reproduced using the examples at the Poisson_CCD code site at [15].

3.1 Pixel distortions and pixel-pixel covariances

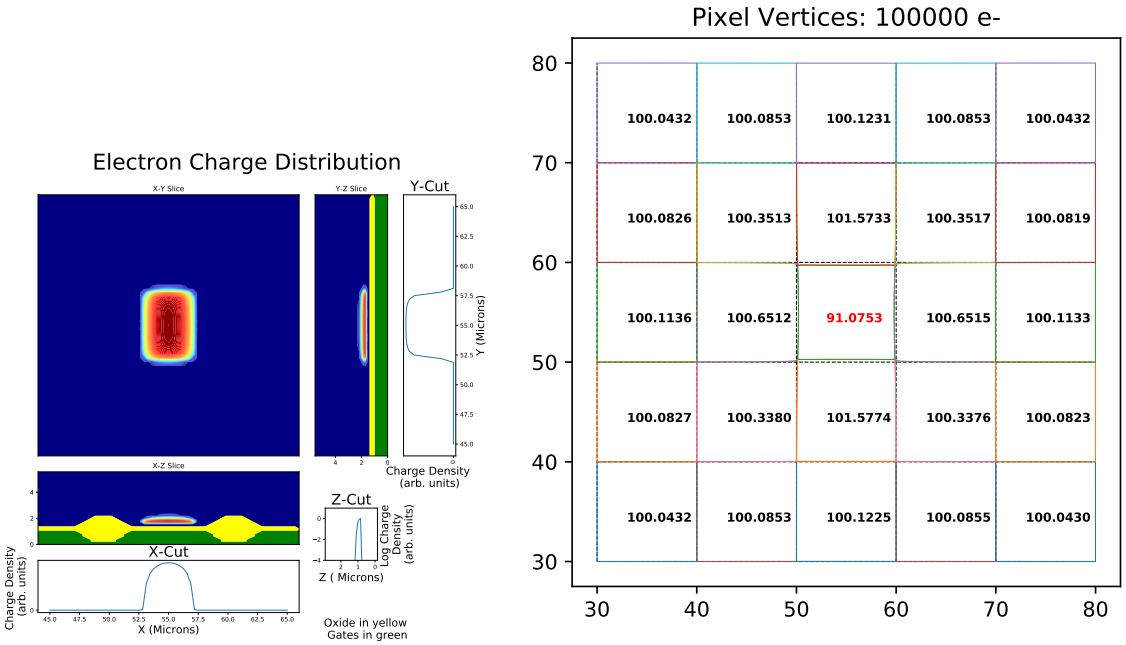
As has been extensively discussed in the literature ([9], [10], [11], [12]), as charge builds up in the central region of bright objects, the stored charge repels additional incoming charge and broadens the profile of these objects. The impact of the stored charge on the pixel shapes can be measured by measuring the pixel-pixel covariances on a large number of flat images ([9], [13]). These covariances are calculated from a large number of flat pairs of varying intensity (see [13] for example) as:

$$C_{i,j} = \frac{\sum_{I,J}(f_{I,J} - \bar{f})(f_{I+i,J+j} - \bar{f})}{\bar{f}^2(N_{pix} - 1)} \tag{17}$$

where $f_{i,j}$ is the difference in flux between the two flats at pixel i,j, and N_{pix} is the number of pixels summed over.

We have generated this data on a large number of flat pairs on LSST CCDs, measured on the UC Davis LSST beam simulator ([22], [12]), and would like to compare these results to simulations. If I take the example of 100 flat pairs, each with 16 million pixels with an average signal level of 50,000 electrons, this is approximately 10^{14} electrons. Directly simulating this data is out of the question. However, we have found a simple way to

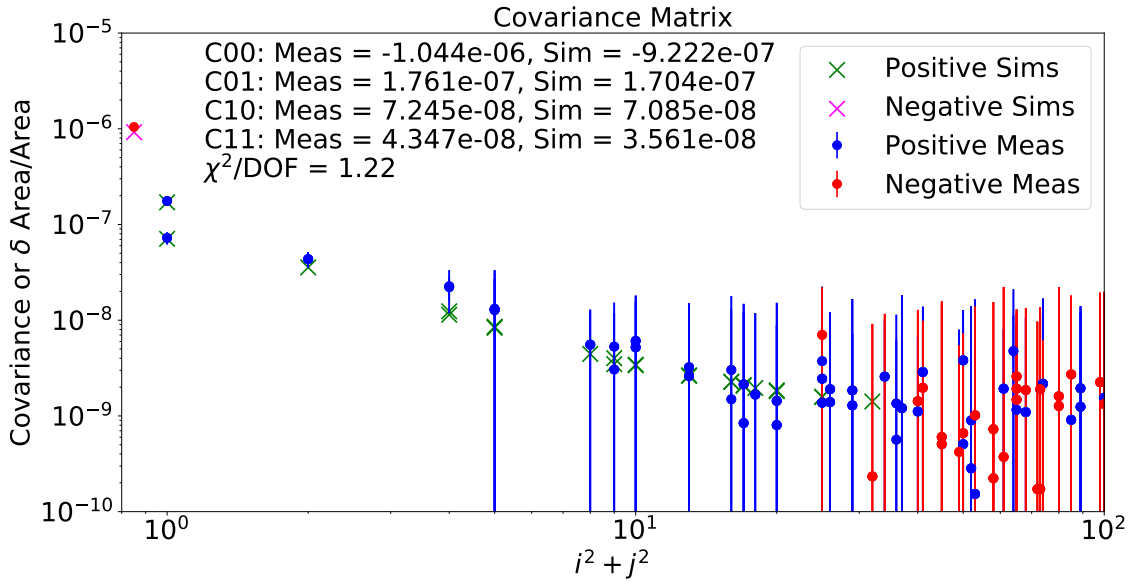
run a single simulation which reproduces the measured pixel covariances. In order to do this, we first simulate a situation where one pixel has a fixed amount of charge (typically 100,000 electrons), and all surrounding pixels are empty. After solving for the potential and resulting electric field, we can track electrons down through the silicon. As the electrons travel down through the silicon under the influence of the electric field, they eventually end up in one of the collecting wells. A binary search is used to find the bifurcation points where electrons on one side of the bifurcation point end up in one pixel, and electrons on the other side of the bifurcation point end up in an adjacent pixel. This binary search is performed with diffusion turned off in order not to introduce a stochastic element into the electron paths. These bifurcation points are identifed as the pixel boundaries. This allows us to characterize the distortion in the pixel boundaries which results from the central pixel charge. A typical result of this process is shown in Figure 12. The distorted pixel shapes which result are what gives rise to the BF effect, with the central pixel losing area, which is gained by the surrounding pixels.



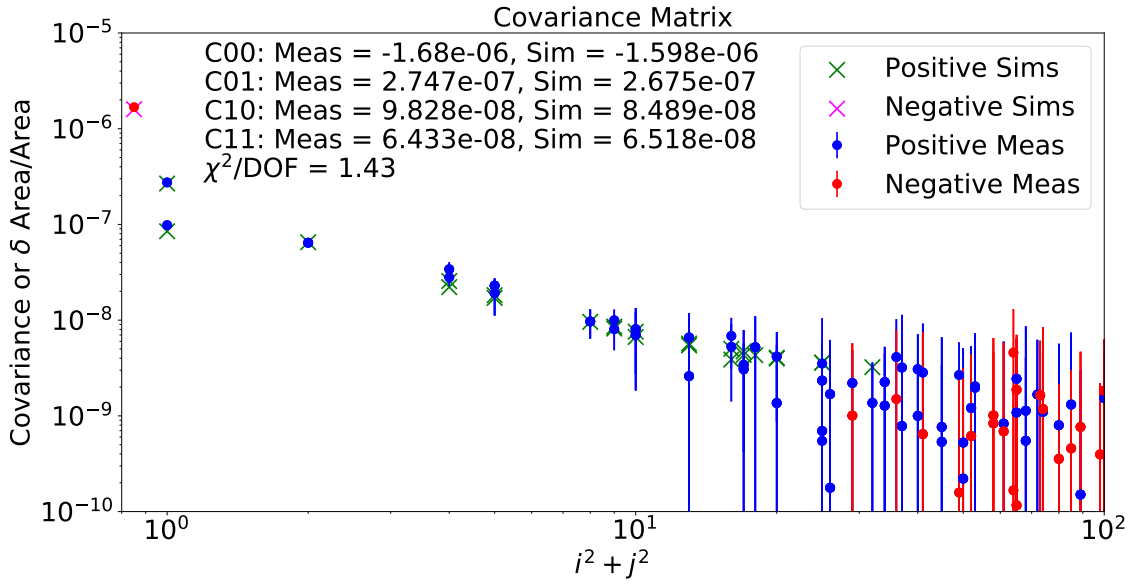
(a) Charge packet with 100,000 electrons (b) Pixel distortion from the central charge packet

Figure 12: Simulation of pixel distortions in an ITL chip when the central pixel contains 100,000 electrons and the surrounding pixels are empty. X and Y are the lateral dimensions of the CCD, and Z is the thickness dimension. The CCDs are 100 microns thick. These distortions are obtained by solving Poisson's equation for the potentials in the CCD, then tracking electrons down and using a binary search to determine the pixel boundaries. As expected, the central pixel loses area and the surrounding pixels all gain area. Note that the loss in area of the central pixel is greater than the sum of the area gains of the surrounding pixels because there are more distant pixels which are not plotted here and which also gain area.

We find that the area distortions which result accurately capture the measured pixel-pixel covariances. Figure 13 shows the agreement between the measured pixel-pixel covariances on flat field images and the simulated area distortions, as measured and as simulated on LSST CCDs from both CCD vendors. The agreement is quite good. The asymmetry of the nearest neighbor pixels is correctly modeled, and the simulated values agree with the measurements within the statistical errors. These simulations are run with the "pixel-itl.cfg" and "pixel-e2v.cfg" examples at [15].



(a) ITL Detector



(b) E2V Detector

Figure 13: Covariance measurements and simulations. The simulated pixel area distortions (see Figure 12) accurately determine the measured pixel-pixel covariances as measured on flat pairs. The circles are the measured covariances, as extracted by the code in the LSST image reduction pipeline as described in the text. The crosses are the fractional area distortions as simulated by the Poisson_CCD code and shown in Figure 12. The leftmost point (the central pixel) has been shifted to an X-axis value of 0.8 to allow plotting it on this log-log plot. Both the E2V and ITL simulations have been informed by physical analysis of both chips, including SIMS dopant profiling and measurements of physical dimensions [14]. Both the covariance measurements and the simulations have been normalized to the distortion caused by one electron. The asymmetry of the nearest neighbor pixels is correctly modeled, and the simulated values agree with the measurements within the statistical errors.

3.2 Diffusion modeling and Fe55 tests

CCDs are routinely characterized by exposing the CCD to an Fe^{55} source. (see [23], for example). The radioactive decay produces X-rays with a known energy, with the K^{α} peak being the strongest peak, typically producing 1620 hole-electron pairs when photoelectrically absorbed in the silicon. These carriers then propagate down to the collecting wells, where they are collected and counted. Because of diffusion, the carriers, which are initially produced in a small volume, spread out and occupy several pixels. To simulate these events, a special module was written. Normally photoelectrons propagate one at a time, without influence from neighboring carriers. But the carriers produced in the Fe^{55} even are produced in a short time, so interactions between the carriers might be important. The code takes the like carrier repulsion and opposite carrier attraction into account, and the parameters "Fe55ElectronMult" and "Fe55HoleMult" can be used to turn off or modify this interaction if desired. Figure 14 shows a typical event, and Figure 15 shows stacked pixel maps compared between measurements and simulations. The spread of the charge cloud due to diffusion is well modeled. This simulation is run with the "fe55.cfg" example at [15].

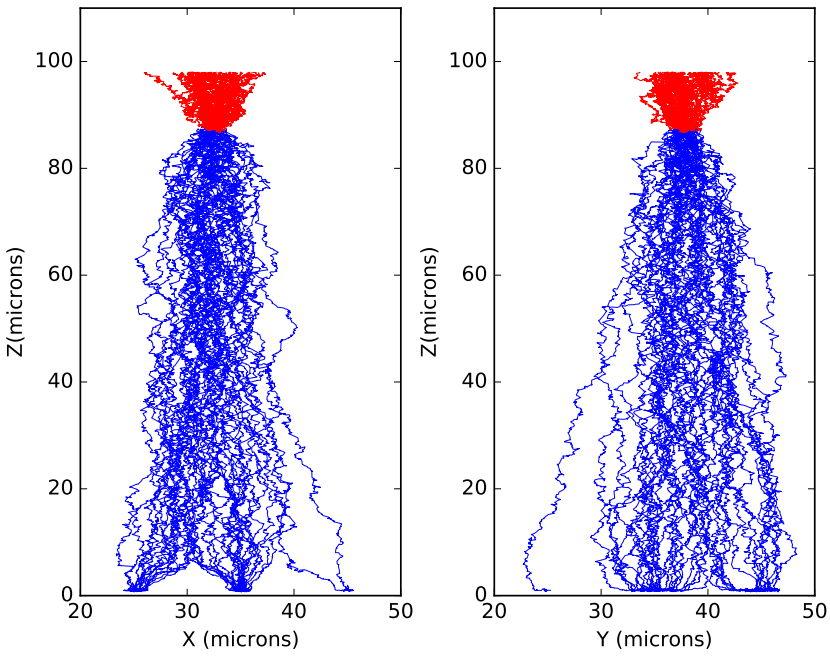
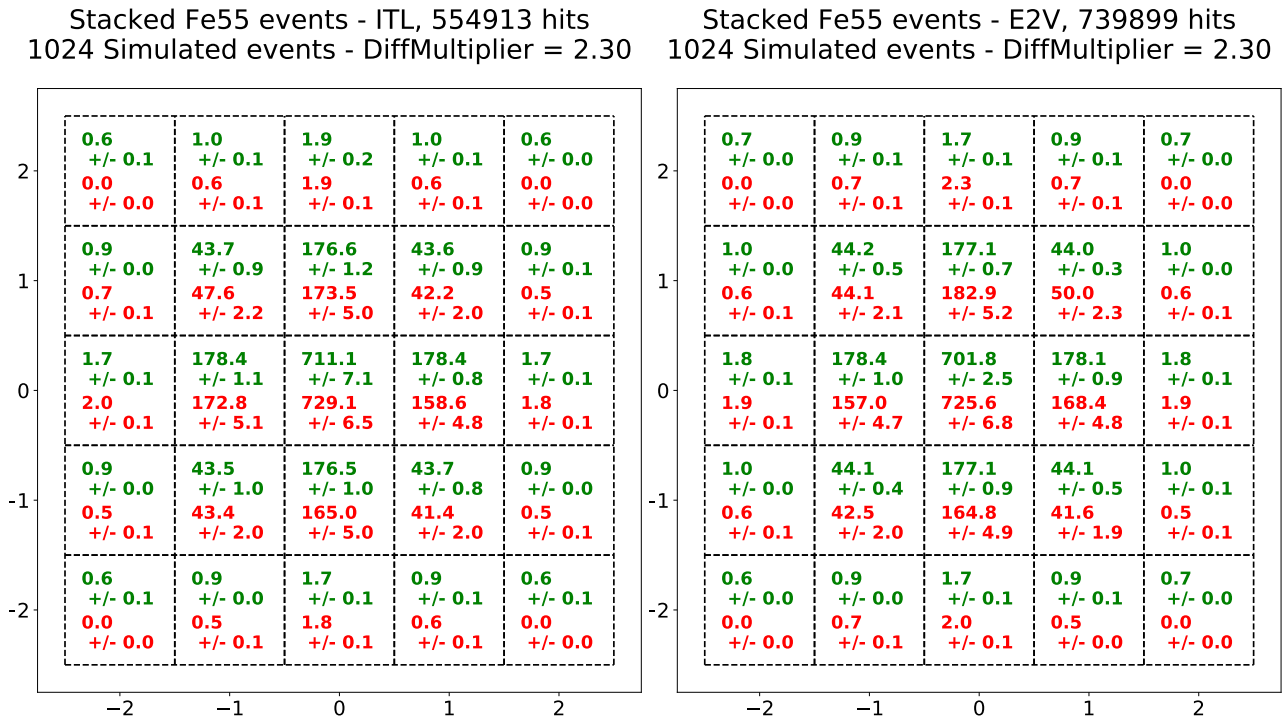


Figure 14: This shows a simulation of a typical Fe^{55} event in an ITL device. Electrons are shown in blue (moving downward) and holes in red(moving upward). The left panel is a slice in the serial direction, and the right panel is a slice in the parallel direction. Only a small fraction of the 1620 incident hole-electron pairs are plotted here.



(a) ITL device - $\chi^2/\mathrm{DOF} = 1.4$ (b) E2V device - $\chi^2/\mathrm{DOF} = 2.1$

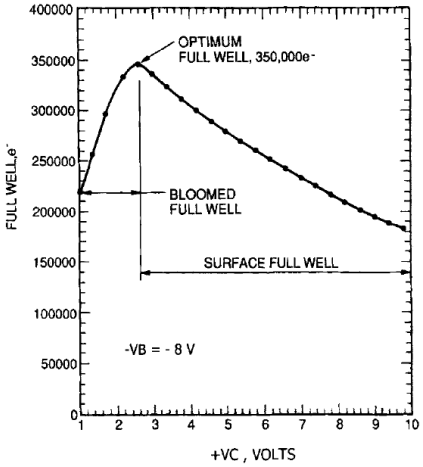
Figure 15: Comparison of Fe^{55} event stacked pixel maps between measurement and simulation for both ITL and E2V devices. The numbers in each pixel are the average number of electrons in each pixel, when the event is centered on the center pixel(2,2). The measurements (top, in green) are a stack of several hundred thousand events, and the errors of the measurements are one sigma values of the 16 amplifiers on one CCD. The simulations (bottom, in red) are a stack of 1024 simulated events, and the errors are statistical. The spread of the charge cloud due to diffusion is well modeled.

3.3 Saturation and blooming

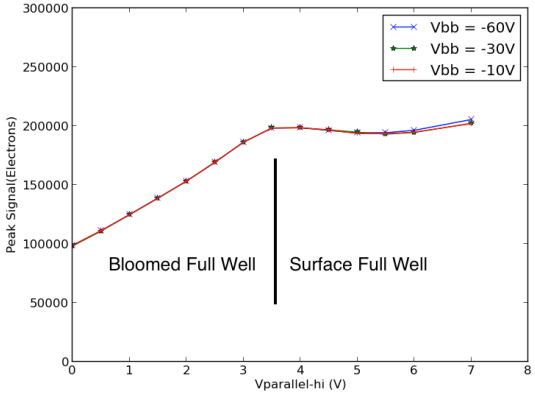
A large number of simulations have been run to better understand saturation and blooming in the ITL STA3800C device. These simulations have been very helpful to understand the physics of the device, because in the simulator one can get information which is simply not accessible to measurement. Figure 16 shows the distinction between the "bloomed full well" condition, where charge begins to bloom above the charge storage regions, and the "surface full well" condition, where charge blooms along the silicon surface. The simulations in Figures 17 and 18 reproduce these conditions, and these simulations illustrate the difference between these conditions. Measurements of saturated spots in the surface full well condition, shown in Figure 19 also show that in the surface full well condition, charge is lost to traps at the silicon-silicon dioxide interface. Thus it is apparent that the surface full well condition is to be avoided.

We can go further and quantitatively reproduce measurements of the onset of saturation as a function of parallel low and high voltages, as shown in Figure 20. By quantifying the barrier height between the storage wells, we show that saturation occurs when the barrier height drops below a certain value. The fit is good except in the strong surface full well condition, because the charge loss that occurs is not included in the simulations. It would be possible to modify the simulations to include this effect, but since the surface full well condition is to be

avoided, this was deemed to be not worth the effort.



(a) Surface full well vs bloomed full well. Reproduced from [23].



(b) Similar measurements on the ITL STA3800C

Figure 16: As discussed in Janesick [23], depending on the parallel gate high voltage, saturation can occur either at the silicon surface or above the collecting wells. This is illustrated in Figures 17 and 18.

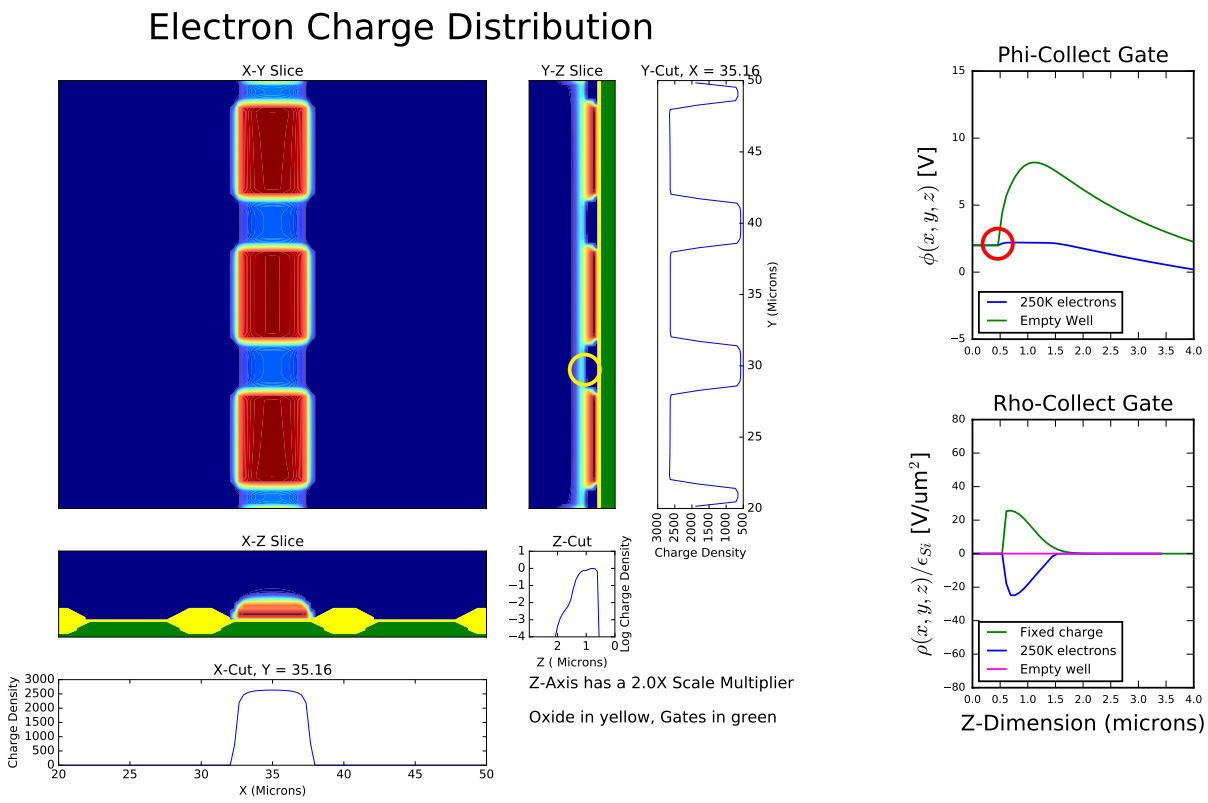


Figure 17: Simulation of the bloomed full well condition. Each of the three pixels contains 250,000 electrons, and the parallel high voltage is 2.0V. The yellow circle shows where charge is blooming above storage wells. The red circle shows that the potential in the storage well is still above that at the gate interface, keeping charge away from the surface.

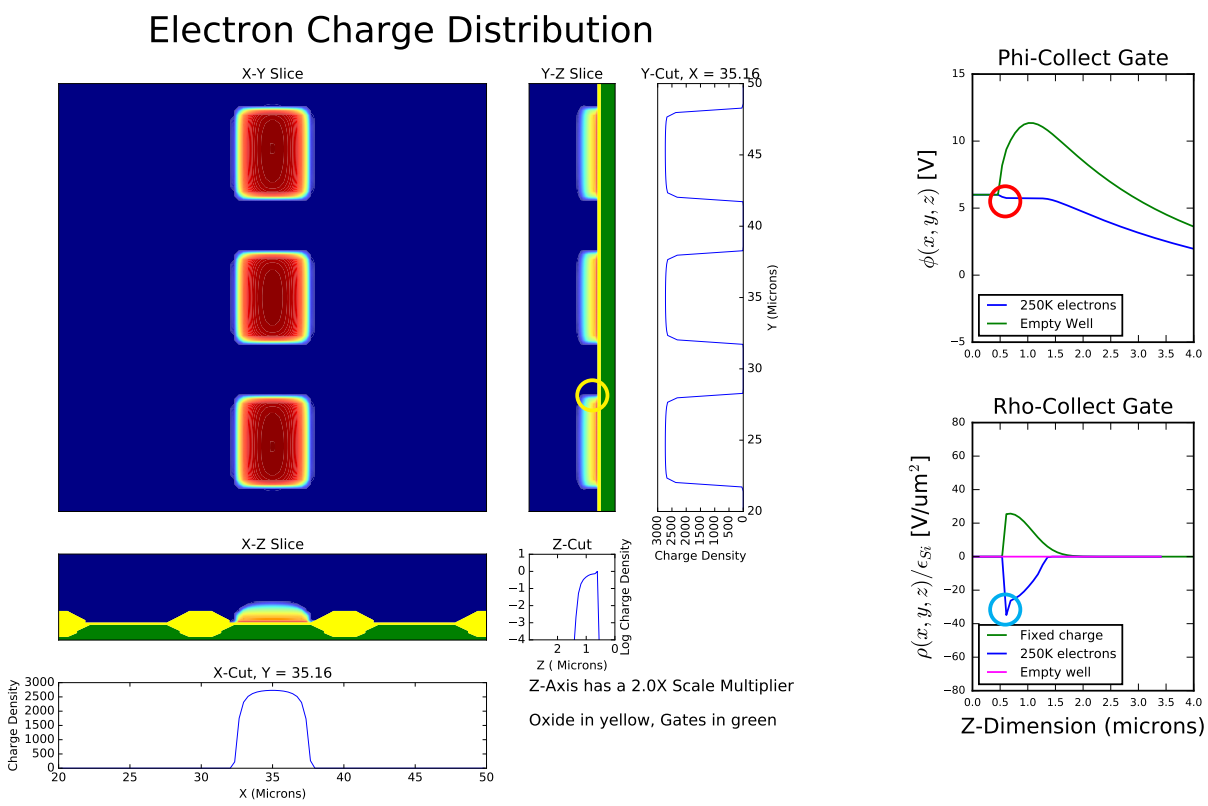
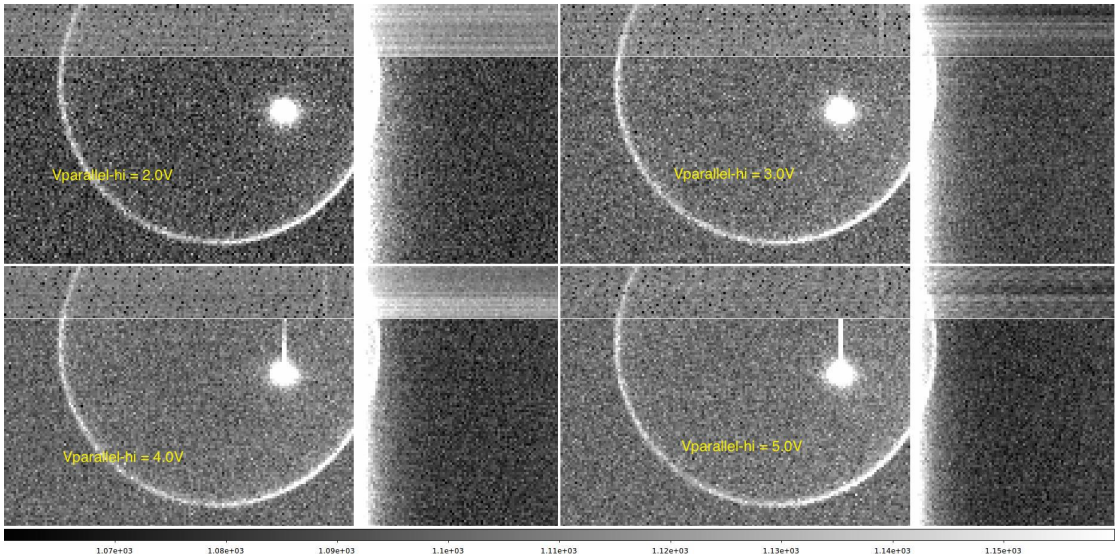
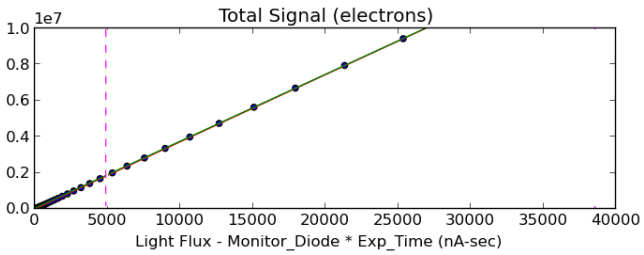


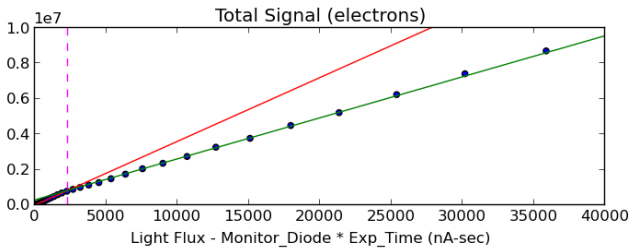
Figure 18: Simulation of the surface full well condition. Each of the three pixels contains 250,000 electrons, and the parallel high voltage is 6.0V. The yellow circle shows where charge is blooming along the silicon surface. The red circle shows that the potential in the storage well is now below that at the gate interface, causing added charge to be added at the surface. The cyan circle shows the surface spike of added charge.



(a) Spot images at different parallel high voltages.

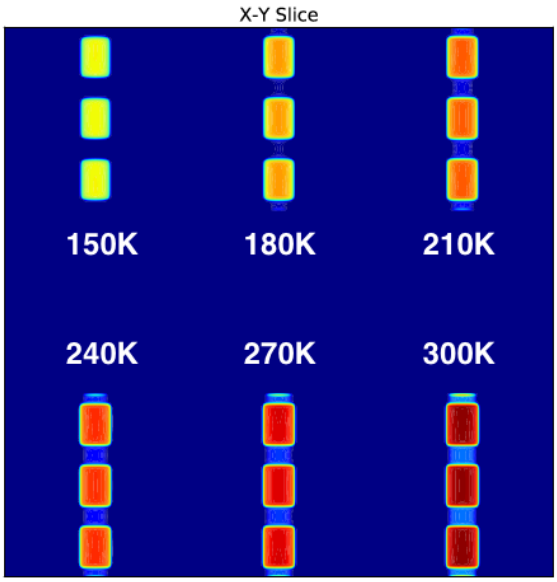


(b) Parallel high voltage of 2.5V - bloomed full well.

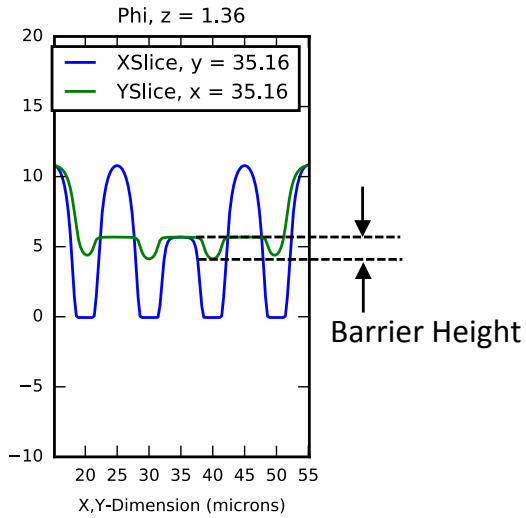


(c) Parallel high voltage of 6.0V - surface full well.

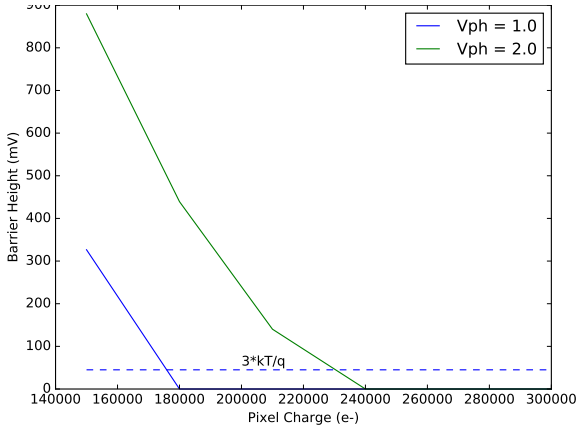
Figure 19: Spot images in the bloomed full well and surface well conditions. In the top panel, we see that we begin to see "trailing" when we enter the surface full well condition, because charges are trapped at the silicon surface. In the bottom two panels, the vertical dotted line indicates the onset of saturation. In the bottom left panel, we see that in the bloomed full well condition the total charge in the saturated spot increases linearly with flux and no charge is lost. In the bottom right panel, we see that in the surface full well condition, charge begins to be lost as we enter saturation. We believe this charge recombines at surface traps at the silicon-silicon dioxide interface.



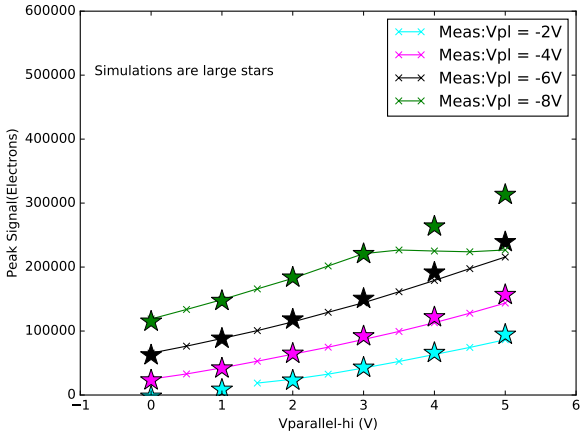
(a) Simulation with increasing charge, showing onset of blooming.



(b) Quantification of barrier height.



(c) Barrier height as a function of pixel charge.



(d) Measurements and simulations of onset of saturation.

Figure 20: Measurements and simulations of the onset of saturation as a function of parallel low and high voltages. Panel (a) shows the simulations which are run, where the pixel charges are increased until saturation is seen. From these simulations, the barrier height between pixels is quantified as a function of pixel charge, as shown in panels (b) and (c). Panel (d) shows that this methodology accurately reproduces measurements of the onset of saturation over most of the range. The simulation fails when we enter the strong surface full well condition, where there is significant charge loss.

3.4 Astrometric shifts at array edges

In addition to simulations of the pixel arrays, simulations can be done of the peripheral circuitry, as shown in the next two sections. At the edges of the pixel array, lateral electric fields from the surounding circuitry can introduce pixel boundary shifts, which lead to measurable astrometric shifts. This effect has been characterized

on the UC Davis LSST Optical Simulator ([22], [24]). We have then built a simulation of a portion of the pixel array which extends to the chip edge to compare the measurements to the simulations. In addition to the pixel array, it is necessary to build into the simulation the appropriate "fixed voltage regions" at the edge of the chip. Figure 21 shows the setup of this simulation. The bending of the equipotential lines near the edge of the pixel array betray the presence of a lateral electric field which deflects incoming electrons. Figure 22 shows these simulated paths (with diffusion turned off), and the edge deflection is apparent. In addition to this deflection, there is a second effect which affects the astrometric shift, which is that as the measured spot begins to "fall off' the edge of the array, there is a resulting shift in the opposite direction. These two competing effects lead to the astrometric shifts seen in Figure 23. The shift has been characterized for a number of different measurement conditions. The simulation, while not perfect, captures all of the trends correctly. These simulations are run with the "edge.cfg" example at [15].

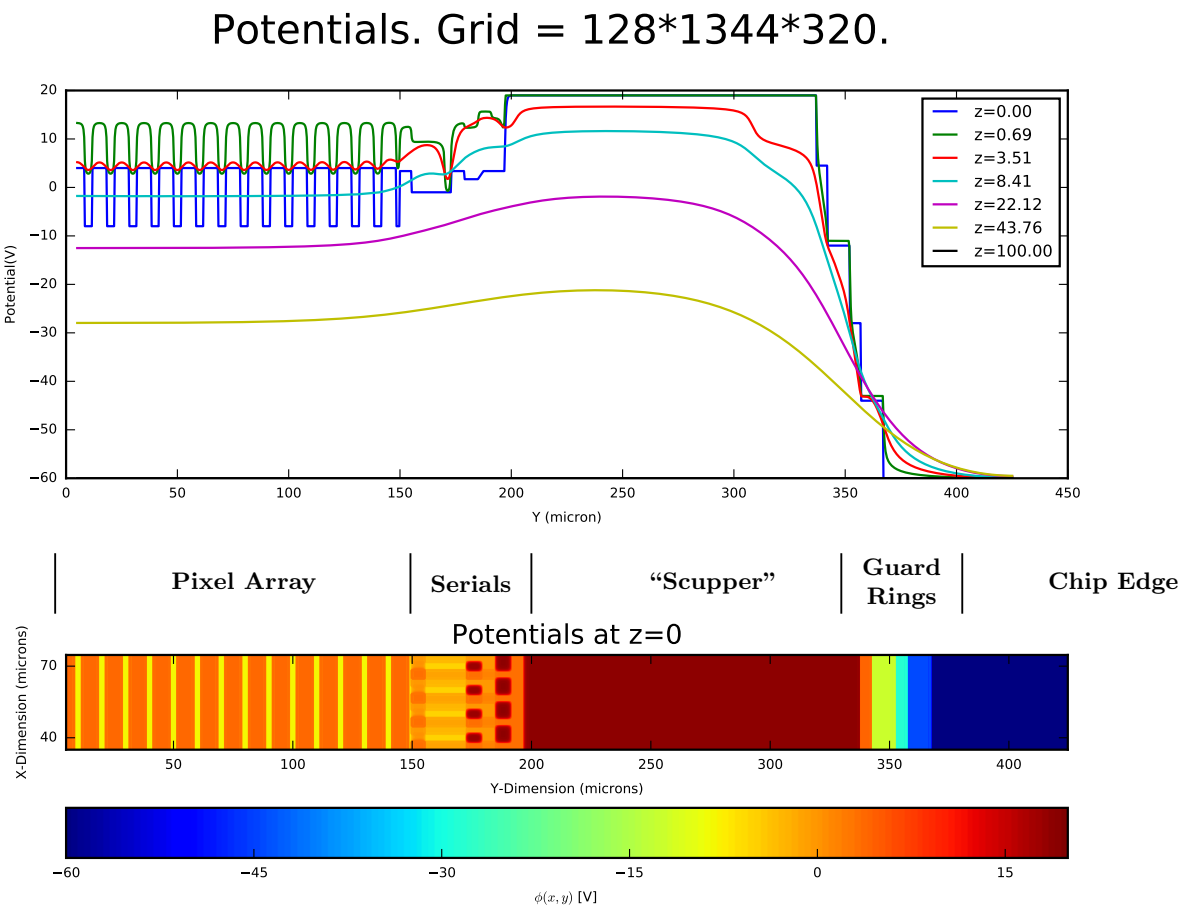


Figure 21: This shows the basic simulation which is run to determine the astrometric shifts at the array edge. The simulation includes a narrow strip of pixels and continues until the edge of the chip.

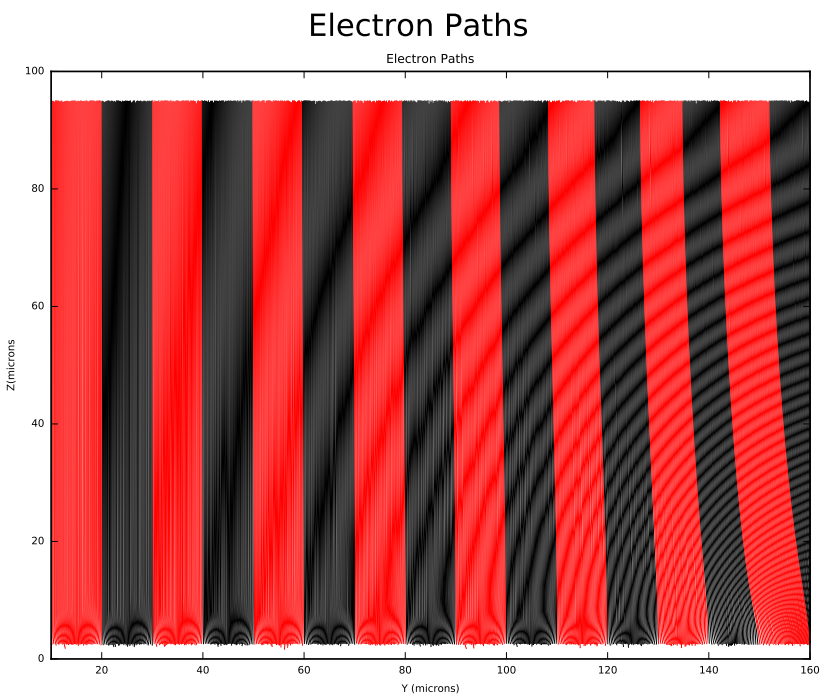


Figure 22: This shows the deviation of electron paths near the edge of the chip due to the lateral electric fields. Diffusion is off for this plot

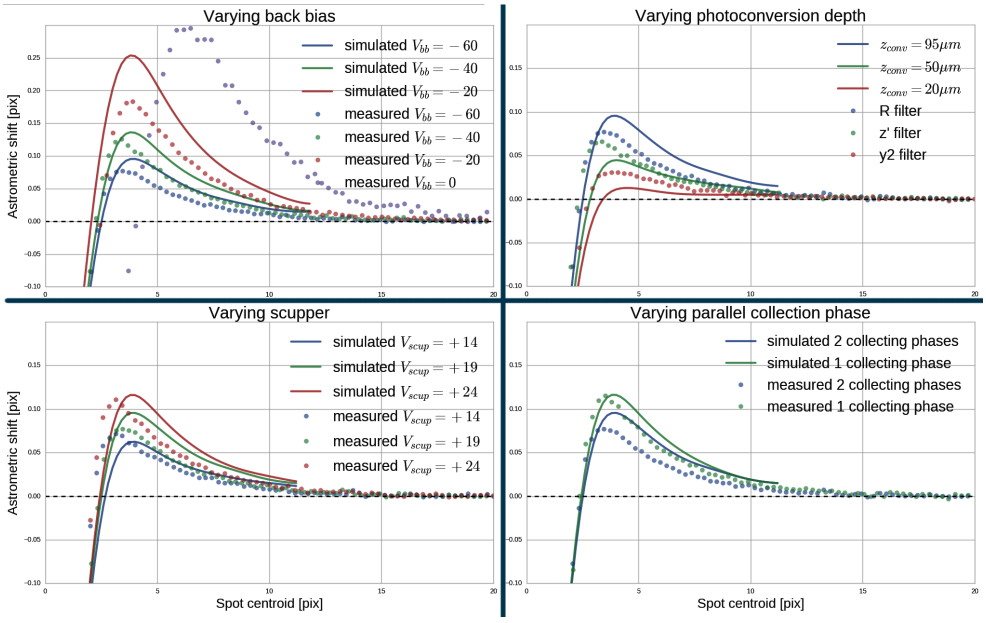
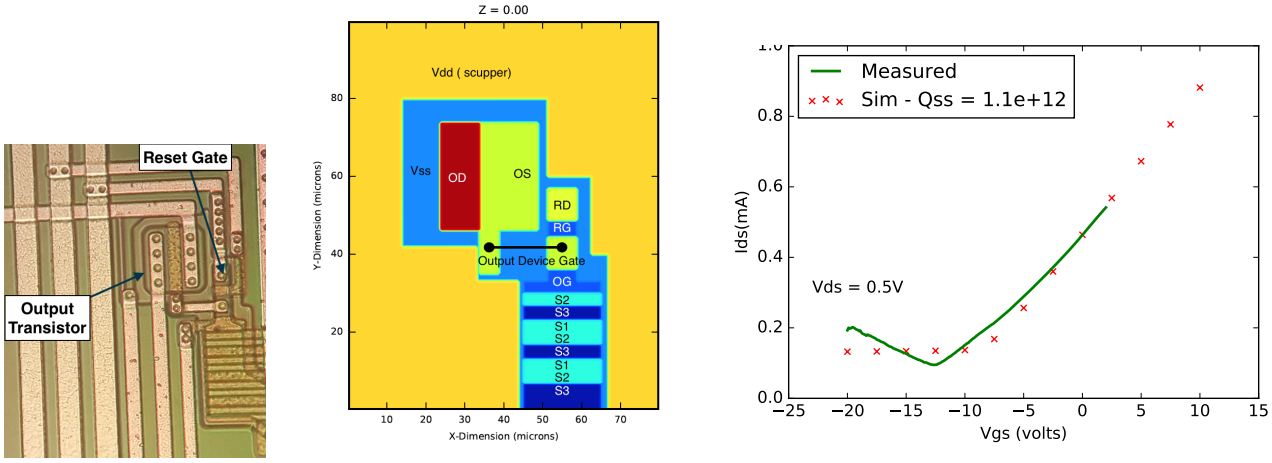


Figure 23: This shows the measured and simulated astrometric shifts at the array edge, with several parameters varied. The simulation captures the major trends of all of these variables. The simulation was not run for Vbb=0, where the chip is not fully depleted.

3.5 Output transistor characteristics

This section shows simulations that were done to model the output transistor of the ITL STA3800C. Figure 24 shows the simulated and measured $I_d - V_g$ characteristics. We obtained a relatively good fit of the transistor turn-on. This simulation is run with the "trans.cfg" example at [15].

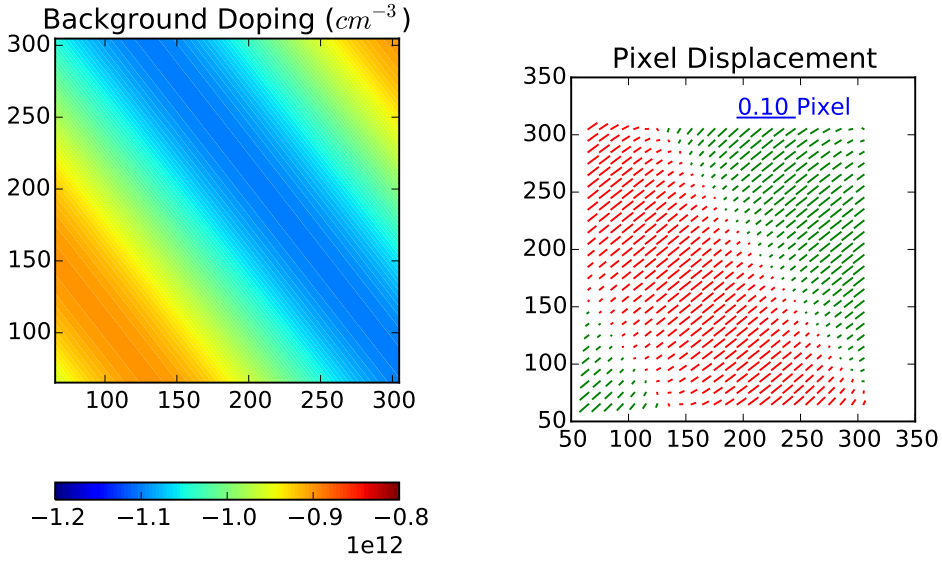


(a) Photograph of the output circuitry (b) Simulation of the same region (c) Simulated and measured I-V

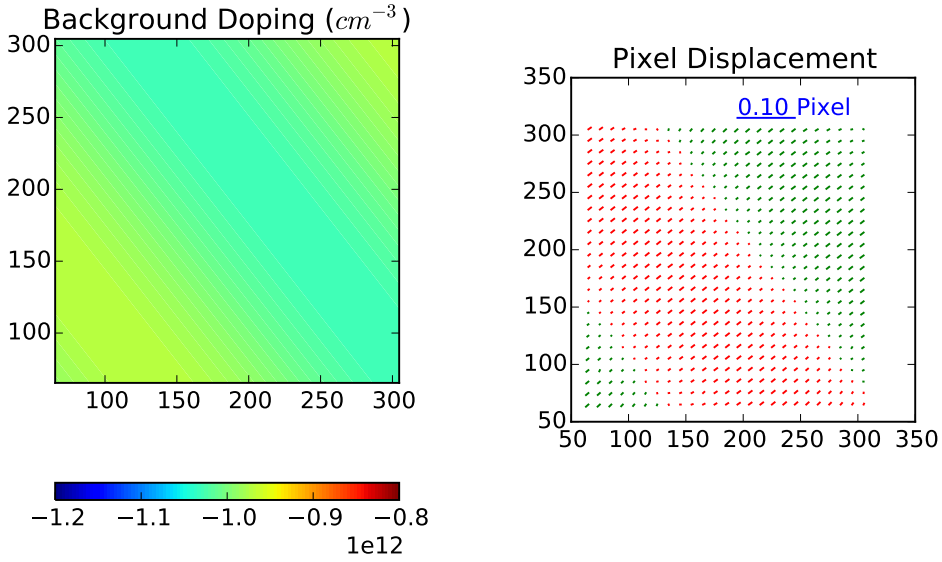
Figure 24: Simulation of the ITL STA3800C output transistor I-V characteristics, compared to measurements.

3.6 Other qualitative tests

Some other tests have been run which have given results which are qualitatively reasonable, but which have not been compared with quantitative measurements, and three of these are reviewed here. The first of these are known as "tree rings". As is well known, periodic dopant variations introduced during the growth of the silicon boule can lead to measurable variations in flat fields, as well as introducing astrometric pixel shifts (see, for example, [8]). This effect has been successfully simulated, as shown in Figure 25. This simulation is run with the "treering.cfg" example at [15].



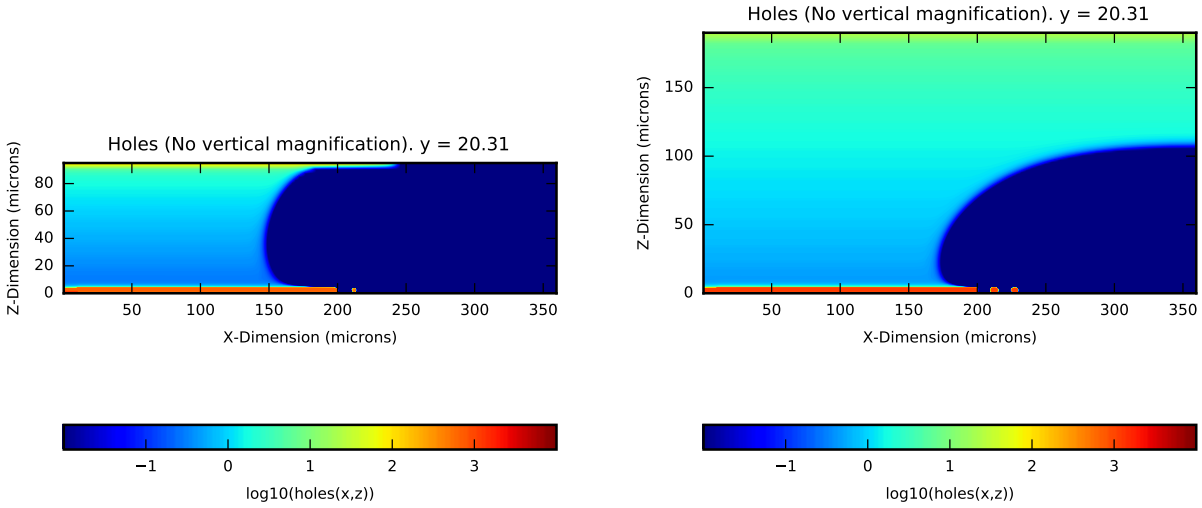
(a) Simulated tree rings with 10% dopant variation



(b) Simulated tree rings with 3% dopant variation

Figure 25: Simulations of "tree rings". A sinusoidal dopant variation in the silicon bulk is introduced, varying in X and Y, and constant in Z. The resulting astrometric pixel shifts are characterized. The X and Y axes are in pixels. The lower value is more consistent with actual observations.

A second interesting simulation is of the backside substrate connection V_{BB}. It is often questioned how this bias voltage, which is only connected to the CCD frontside, is conducted to the backside when the CCD is fully depleted. The answer is that there is an undepleted region near the chip edge which serves this purpose. It is interesting to simulate this connection, and the result is shown in Figure 26.



(a) $T_{Si} = 100\mu m$; $V_{BB} = -60V$ (b) $T_{Si} = 200\mu m$; $V_{BB} = -9V$

Figure 26: Simulations of the V_{BB} guard rings for two different conditions. In both cases, the top of the simulation region is where photons are incident, and the left hand edge is the edge of the CCD. The imaging region begins at the right hand edge of the simulation region and continues to the right. The colors are the log of the hole concentration (in code units), as shown in the colorbar. The imaging region in the left hand simulation is fully depleted, while in the right hand one, which is thicker and has a lower bias voltage, it is not.

While the simulator solves for the condition of the CCD in equilibrium, and does not do transient simulations, it is possible to simulate transient effects by repeatedly solving for the state of the CCD with small incremental changes to the boundary conditions. These can then be stitched together to form a movie of the results. An example of the parallel charge transport is shown in Figure 27. Several movies constructed in this way are available at [15].

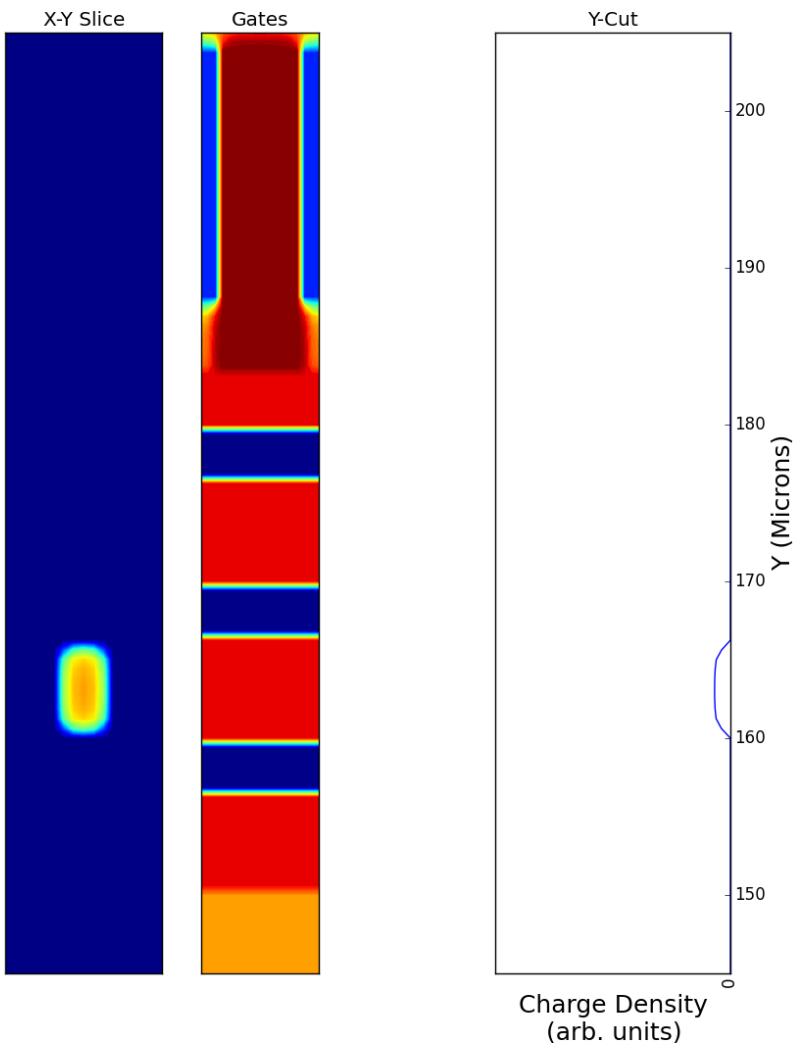


Figure 27: A single frame of a movie showing the transport of a charge packet. The left panel shows the charge distribution, the center panel shows the surface potential, and the right panel shows a 1D slice through the charge distribution. In the actual movie (available at [15]), the charge packet can be seen propagating through the parallel chain and into the serial register.

4 Conclusion

We have presented a software package optimized for simulating astronomical CCDs. The code has been optimized to be as physically relaistic as possible, and accurately simulates many aspects of CCD behavior. The package is open source and freely available, and has been validated against a number of different types of CCD measurements. It has proven useful for analyzing sensor effects in the CCDs being used to construct the LSST focal plane, and insights gained from running these simulations are being incorporated into the software stack to be used for instrument signature removal in the LSST images. It is hoped that it will find additional uses in the future.

5 Acknowledgements

Kirk Gilmore's help in setting up the hardware and software for the CCDs from both vendors has also been invaluable. Perry Gee has provided key support in software and networking. Merlin Fisher-Levine has patiently helped with my understanding and implementation of the LSST data management software stack. David Kirkby contributed code for including different filters. Financial support from DOE grant DE-SC0009999 and Heising-Simons Foundation grant 2015-106 are gratefully acknowledged.

References

[1] George E. Smith. Nobel lecture: The invention and early history of the ccd. *Rev. Mod. Phys.*, 82:2307–2312, Aug 2010.

[2] Ž. Ivezić, S. M. Kahn, J. A. Tyson, B. Abel, E. Acosta, R. Allsman, D. Alonso, Y. AlSayyad, S. F. Anderson, J. Andrew, and et al. LSST: From Science Drivers to Reference Design and Anticipated Data Products. *Astrophys. J.*, 873:111, March 2019.

[3] P. O'Connor. Uniformity and Stability of the LSST Focal Plane, 2019. arXiv:1907.00995.

[4] P. O'Connor, P. Antilogus, P. Doherty, J. Haupt, S. Herrmann, M. Huffer, C. Juramy-Giles, J. Kuczewski, S. Russo, C. Stubbs, and R. Van Berg. Integrated system tests of the LSST raft tower modules. In Andrew D. Holland and James Beletic, editors, *High Energy, Optical, and Infrared Detectors for Astronomy VII*, volume 9915, pages 327 – 338. International Society for Optics and Photonics, SPIE, 2016.

[5] University of Arizona. Imaging Technology Laboratory, 2016. http://www.itl.arizona.edu.

[6] Teledyne E2V, 2019. https://www.teledyne-e2v.com/products/space/.

[7] J. Villasenor, G. Prigozhin, J.P. Doty, C. Lage, S. Yazdi, G. Ricker, and R. Vanderspek. Reach-through effect in deep depletion TESS CCDs. *Journal of Instrumentation*, 12(04):C04025–C04025, apr 2017.

[8] A. A. Plazas, G. M. Bernstein, and E. S. Sheldon. On-Sky Measurements of the Transverse Electric Fields' Effects in the Dark Energy Camera CCDs. *Pub. Astron. Soc. Pacific*, 126(942):750, Aug 2014.

[9] P. Antilogus, P. Astier, P. Doherty, A. Guyonnet, and N. Regnault. The brighter-fatter effect and pixel correlations in CCD sensors. *Journal of Instrumentation*, 9:C03048, March 2014.

[10] D. Gruen, G. M. Bernstein, M. Jarvis, B. Rowe, V. Vikram, A. A. Plazas, and S. Seitz. Characterization and correction of charge-induced pixel shifts in DECam. *Journal of Instrumentation*, 10:C05032, May 2015.

[11] A. Guyonnet, P. Astier, P. Antilogus, N. Regnault, and P. Doherty. Evidence for self-interaction of charge distribution in charge-coupled devices. *Astro. & Astrophys.*, 575:A41, March 2015.

[12] C. Lage, A. Bradshaw, and J. A. Tyson. Measurements and simulations of the brighter-fatter effect in CCD sensors. *Journal of Instrumentation*, 12:C03091, Mar 2017.

[13] William R. Coulton, Robert Armstrong, Kendrick M. Smith, Robert H. Lupton, and David N. Spergel. Exploring the Brighter-fatter Effect with the Hyper Suprime-Cam. *The Astronomical Journal*, 155(6):258, May 2018.

[14] C. Lage. Physical and electrical analysis of LSST sensors, 2019. In preparation.

[15] C. Lage. Poisson solver for LSST CCDs, November 2019. https://github.com/craiglagegit/Poisson_CCD.

[16] S.M. Sze. *Physics of Semiconductor Devices*. Wiley-Interscience publication. John Wiley & Sons, 1981.

[17] Conor S Rafferty, Mark R Pinto, and Robert W Dutton. Iterative methods in semiconductor device simulation. *IEEE Transactions on Electron Devices*, 32(10):2018–2027, 1985.

[18] William L Briggs, Steve F McCormick, et al. *A multigrid tutorial*, volume 72. Siam, 2000.

[19] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes*. Cambridge University Press, third edition, 2007.

[20] Martin A. Green. Intrinsic concentration, effective densities of states, and effective mass in silicon. *Journal of Applied Physics*, 67(6):2944–2954, 1990.

[21] C. Jacobini, C. Canali, G.. Ottaviani, and A. Quaranta. A review of some transport properties of silicon. *Solid-State Electronics*, 20:77–89, 1977.

[22] J. A. Tyson, J. Sasian, K. Gilmore, A. Bradshaw, C. Claver, M. Klint, G. Muller, G. Poczulp, and E. Resseguie. LSST optical beam simulator. In *High Energy, Optical, and Infrared Detectors for Astronomy VI*, volume 9154 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 915415, July 2014.

[23] J.R. Janesick. *Scientific Charge-coupled Devices*. Press Monograph Series. SPIE Press, 2001.

[24] A. Bradshaw, C. Lage, E. Resseguie, and J. A. Tyson. Mapping charge transport effects in thick CCDs with a dithered array of 40,000 stars. *Journal of Instrumentation*, 10:C04034, April 2015.

A List of configuration parameters

Name	type	Default	Description
AddTreeRings	int	0	0-No tree rings, 1-Add tree rings
BackgroundDoping	float	-1e+12	Background doping in cm^{-3}
BottomSteps	int	1000	Number of diffusion steps each electron takes while logging final charge location
BuildQFeLookup	int	0	0-Don't build look-up table, 1-build look-up table
CCDTemperature	float	173.00	Temperature in K
CalculateZ0	int	0	0 - don't calculate - Use ElectronZ0, 1 - calculate from filter and SED.
ChannelDepth	float	1.00	Square profile depth in microns
ChannelDoping	float	5e+11	Square profile doping in cm^{-3}
ChannelDose	float	5e+11	Gaussian profile dose in cm^{-2}
ChannelPeak	float	0.00	Gaussian peak depth in microns
ChannelProfile	int	0	0 = Square profile, N = N Gaussian profiles
ChannelSigma	float	0.50	Gaussian sigma in microns
ChannelStopDepth	float	1.00	Square profile depth in microns
ChannelStopDoping	float	5e+11	Square profile doping in cm^{-3}
ChannelStopDose	float	5e+11	Gaussian profile dose in cm^{-2}
ChannelStopDotCenter	float	5.00	Center position in microns from pixel bottom
ChannelStopDotDepth	float	1.00	Square profile depth in microns
ChannelStopDotDoping	float	5e+11	Square profile doping in cm^{-3}
ChannelStopDotDose	float	5e+11	Gaussian profile dose in cm^{-2}
ChannelStopDotHeight	float	0.00	Height in microns
ChannelStopDotPeak	float	0.00	Gaussian peak depth in microns
ChannelStopDotProfile	int	0	0 = Square profile, N = N Gaussian profiles
ChannelStopDotSigma	float	0.50	Gaussian sigma in microns
ChannelStopDotSurfaceCharge	float	0.00	Surface charge in cm^{-2}
ChannelStopPeak	float	0.00	Gaussian peak depth in microns
ChannelStopProfile	int	0	0 = Square profile, N = N Gaussian profiles
ChannelStopSideDiff	float	FieldOxideTaper	Side diffusion in microns
ChannelStopSigma	float	0.50	Gaussian sigma in microns
ChannelStopSurfaceCharge	float	0.00	Surface charge in cm^{-2}
ChannelStopWidth	float	1.00	Width in microns
ChannelSurfaceCharge	float	0.00	Surface charge in cm^{-2}
CollectedCharge[i][j]	int	0	Number of electron in well i,j
CollectingPhases	int	1	Number of collecting phases
Continuation	int	0	0-No continuation, 1-continue at LastCont..Step
DiffMultiplier	float	2.30	Used to adjust amount of diffusion
ElectronMethod	int	0	Controls electron calculation 0 - Leave electrons where they land from tracking (old) 1 - Set QFe (QFe is always used in Fixed Regions) If 1 is specified, you must provide QFe lookup params 2 - Electron conservation and constant QFe (best)
ElectronZ0Area	float	100.00	Initial Z value for electrons calculating pixel areas
ElectronZ0Fill	float	100.00	Initial Z value for electrons filling pixel wells
EquilibrateSteps	int	100	Number of diffusion steps each electron takes after reaching bottom and before beginning to log charge.

Name	type	Default	Description
Fe55CloudRadius	float	0.20	Cloud radius in microns
Fe55ElectronMult	float	1.00	Used to adjust cloud electron attraction/repulsion
Fe55HoleMult	float	1.00	Used to adjust cloud hole attraction/repulsion
FieldOxide	float	0.40	Thickness in microns
FieldOxideTaper	float	0.50	Taper width in microns
FilledPixelCoords[i][j]	float	2.00	Center x,y (in microns) of well i,j
FilterBand	str	none	One of u,g,r,i,z,y
FilterFile	str	notebooks/depth_pdf.dat	SED file
FixedRegionBCType	int	0	Fixed region description
FixedRegionDoping	int	0	Fixed region description
FixedRegionOxide	int	0	Fixed region description
FixedRegionQFe	float	100.00	Fixed region description
FixedRegionQFh	float	-100.00	Fixed region description
FixedRegionVoltage	float	0.00	Fixed region description
FringeAngle	float	0.00	Fringe parameter for PixelBoundaryTestType=3
FringePeriod	float	0.00	Fringe parameter for PixelBoundaryTestType=3
GateGap	float	0.00	Gap between paralel gates in microns (experimental)
GateOxide	float	0.15	Thickness in microns
GridsPerPixelX	int	16	Grids per pixel at ScaleFactor = 1
GridsPerPixelY	int	16	Grids per pixel at ScaleFactor = 1
LastContinuationStep	int	0	Used when continuing a stopped simulation
LogEField	int	0	0 - don't calculate E-Field, 1 - Calc. and store E-Field
LogPixelPaths	int	0	0 - only the final (z 0) point is logged, 1 - Entire path is logged
NQFe	int	81	Number of steps in QFe look-up table
NZExp	float	10.00	Non-linear z axis exponent
NumDiffSteps	int	1	A speed/accuracy trade-off. A value of 1 uses the theoretical diffusion step. A higher value takes larger steps. Experimental
NumElec	int	1000	Number of electrons to be traced between field recalc.
NumPhases	int	3	Number of parallel phases
NumSteps	int	100	Number of steps, each one adding NumElec electrons
NumVertices	int	2	Number of vertices per side for the pixel area calc. Since there are also 4 corners, there will be: (4 * NumVertices + 4) vertices in each pixel
NumberofFilledWells	int	0	Self-explanatory
NumberofFixedRegions	int	0	Self-explanatory
NumberofPixelRegions	int	0	Self-explanatory
Nx	int	160	Number of grids in x at ScaleFactor = 1
Ny	int	160	Number of grids in y at ScaleFactor = 1
Nz	int	160	Number of grids in z at ScaleFactor = 1
Nzelec	int	32	No. grids in z in elec & hole grids at ScaleFactor = 1
PhotonList	str	PhotonList	Photon list filename
PixelAreas	int	0	-1 - Don't calc areas, N - calc areas every Nth step
PixelBoundaryLowerLeft	float	2.00	x,y coordinates of PixelBoundary lower left corner
PixelBoundaryNx	int	9	Number of pixels in PixelBoundary
PixelBoundaryNy	int	9	Number of pixels in PixelBoundary
PixelBoundaryStepSize	float	2.00	Used with PixelBoundaryTestType=0
PixelBoundaryTestType	int	0	0-Trace uniform grid, 1-TraceGaussian spot, 2,4-TraceRegion, 5-TraceList, 6-Fe55 cloud.

Name	type	Default	Description
PixelBoundaryUpperRight	float	2.00	x,y coordinates of PixelBoundary lower left corner
PixelRegionLowerLeft	float	2.00	PixelRegion is used for PBTestType 0,2,4
PixelRegionUpperRight	float	2.00	PixelRegion is used for PBTestType 0,2,4
PixelSizeX	float	-1.00	Pixel size in microns
PixelSizeY	float	-1.00	Pixel size in microns
QFemax	float	10.00	Max QFe in look-up table
QFemin	float	5.00	Min QFe in look-up table
SaturationModel	int	0	Experimental
SaveData	int	1	0 - Save only Pts, N save phi,rho,E every Nth step
SaveElec	int	1	0 - Save only Pts, N save Elec every Nth step
SaveMultiGrids	int	0	0 - Don't save subgrids, 1 - Save all of the grids at all scales
ScaleFactor	int	1	Power of 2 that sets the grid size
Seed	int	77	Pseudo random number seed
SensorThickness	float	100.00	Sensor Thickness in microns
Sigmax	float	1.00	Gaussian spot sigma in microns
Sigmay	float	1.00	Gaussian spot sigma in microns
SimulationRegionLowerLeft	float	2.00	x,y coordinates of lower left corner of entire simulation
TopAbsorptionProb	float	0.00	Probability an electron is absorbed if it reaches the top surface
TreeRingAmplitude	float	0.00	Fractional amplitude (i.e. 0.10 is a 10% variation)
TreeRingAngle	float	0.00	Direction of variation in degrees (0:constant in x)
TreeRingPeriod	float	0.00	Period of sinusoidal variation in microns
Vbb	float	-50.00	Back bias voltage
VerboseLevel	int	1	0 - minimal output, 1 - normal, 2 - more verbose 3-dump everything
Vparallel_lo	float	-8.00	Parallel Low Voltage
Vparallel_hi	float	4.00	Parallel High Voltage
XBCType	int	1	0 - Free BC, 1 - Periodic BC
Xoffset	float	0.00	Shift of Gaussian spot center
YBCType	int	1	0 - Free BC, 1 - Periodic BC
Yoffset	float	0.00	Shift of Gaussian spot center
iterations	int	1	Number of VCycles
ncycle	int	100	Number of SOR cycles at each resolution
outputfilebase	str	Test	Output filename base
outputfiledir	str	data	Output filename directory
qfh	float	-100.00	Hole quasi Fermi level - applies everywhere except Fixed regions
w	float	1.90	Successive Over-Relaxation factor

B Description of example configuration files included with the code.

There are a total of 14 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, and a detailed listing of all configuration parameters is in Appendix A. 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/*/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 1: data/smallpixel/smallpixel.cfg
    1. Purpose: A single pixel and surroundings. The central pixel contains 100,000 electrons. No electron tracking or pixel boundary plotting is done. The subgrids are saved so one can look at convergence. This is useful for getting things set up rapidly

2. Syntax: src/Poisson data/smallpixel/smallpixel.cfg
3. Expected run time: $<$ 1minute.
4. Plot Syntax: python pysrc/Poisson_Small.py data/smallpixel/smallpixel.cfg 0
5. Plot Syntax: python pysrc/Poisson_Convergence.py data/smallpixel/smallpixel.cfg 0

- Example 2: data/pixel0/pixel.cfg

1. Purpose: A 9x9 grid of pixels at low resolution (ScaleFactor=1). The central pixel contains 100,000 electrons. No electron tracking or pixel boundary plotting is done.
2. Syntax: src/Poisson data/pixel0/pixel.cfg
3. Expected run time: $\approx$ 1minute.
4. Plot Syntax: python pysrc/Poisson_Plots.py data/pixel0/pixel.cfg 0
5. Plot Syntax: python pysrc/ChargePlots.py data/pixel0/pixel.cfg 0 2

- Example 3: data/pixel-itl/pixel.cfg

1. Purpose: A 9x9 grid of pixels at higher resolution (ScaleFactor=2). The central pixel contains 100,000 electrons. The parameters are set up for the ITL STA3800C CCD. This should give physically meaningful results, and is what was used in the published papers. After solving Poisson's equation, electron tracking is done to determine the pixel distortions.
2. Syntax: src/Poisson data/pixel-itl/pixel.cfg
3. Expected run time: $\approx$ 30minutes.
4. Plot Syntax: python pysrc/Poisson_Plots.py data/pixel-itl/pixel.cfg 0
5. Plot Syntax: python pysrc/ChargePlots.py data/pixel-itl/pixel.cfg 0 2
6. Plot Syntax: python pysrc/VertexPlot.py data/pixel-itl/pixel.cfg 0 2
7. Plot Syntax: python pysrc/Area_Covariance_Plot.py data/pixel-itl/pixel.cfg 0

- Example 4: data/pixel-e2v/pixel.cfg

1. Purpose: A 9x9 grid of pixels at higher resolution (ScaleFactor=2). The central pixel contains 100,000 electrons. The parameters are set up for the E2V CCD250 CCD. This should give physically meaningful results, and is what was used in the published papers. After solving Poisson's equation, electron tracking is done to determine the pixel distortions.
2. Syntax: src/Poisson data/pixel-e2v/pixel.cfg
3. Expected run time: $\approx$ 30minutes.
4. Plot Syntax: python pysrc/Poisson_Plots.py data/pixel-e2v/pixel.cfg 0
5. Plot Syntax: python pysrc/ChargePlots.py data/pixel-e2v/pixel.cfg 0 2
6. Plot Syntax: python pysrc/VertexPlot.py data/pixel-e2v/pixel.cfg 0 2
7. Plot Syntax: python pysrc/Area_Covariance_Plot.py data/pixel-e2v/pixel.cfg 0

- Example 5: data/pixel1/pixel.cfg

1. Purpose: A 9x9 grid of pixels at higher resolution (ScaleFactor=2). The central pixel contains 100,000 electrons. The parameters are set up for the ITL STA3800C CCD. This is included as an example of ElectronMethod=1, which iterates to find the value of the electron Quasi_Fermi level. After solving Poisson's equation, electron tracking is done to determine the pixel distortions.
2. Syntax: src/Poisson data/pixel1/pixel.cfg
3. Expected run time: $\approx$ 45minutes.
4. Plot Syntax: python pysrc/Poisson_Plots.py data/pixel1/pixel.cfg 0

5. Plot Syntax: python pysrc/ChargePlots.py data/pixel1/pixel.cfg 0 2
6. Plot Syntax: python pysrc/VertexPlot.py data/pixel1/pixel.cfg 0 2
7. Plot Syntax: python pysrc/Area_Covariance_Plot.py data/pixel1/pixel.cfg 0

- Example 6: data/smallcapf/smallcap.cfg
  1. Purpose: A simple field oxide capacitor for testing convergence.
  2. Syntax: src/Poisson data/smallcapf/smallcap.cfg
  3. Expected run time: $\approx$ 1minute.
  4. Plot Syntax: python pysrc/Poisson_CapConvergence.py data/smallcapf/smallcap.cfg 0
- Example 7: data/smallcapg/smallcap.cfg
  1. Purpose: A simple gate oxide capacitor for testing convergence.
  2. Syntax: src/Poisson data/smallcapg/smallcap.cfg
  3. Expected run time: $\approx$ 1minute.
  4. Plot Syntax: python pysrc/Poisson_CapConvergence.py data/smallcapg/smallcap.cfg 0
- Example 8: data/edgerun/edge.cfg
  1. Purpose: A simulation of the pixel distortion at the top and bottom edges of the ITL STA3800C CCD.
  2. Syntax: src/Poisson data/edgerun/edge.cfg
  3. Expected run time: $\approx$ 15minutes.
  4. Plot Syntax: python pysrc/Poisson_Edge.py data/edgerun/edge.cfg 0
  5. Plot Syntax: python pysrc/Pixel_Shift.py data/edgerun/edge.cfg 0
- Example 9: data/iorun/io.cfg
  1. Purpose: A simulation of the output transistor region of the ITL STA3800C CCD, including the last few serial stages.
  2. Syntax: src/Poisson data/iorun/io.cfg
  3. Expected run time: $\approx$ 5minutes.
  4. Plot Syntax: python pysrc/Poisson_IO.py data/iorun/io.cfg 0
  5. Plot Syntax: python pysrc/ChargePlots_IO.py data/iorun/io.cfg 0
- Example 10: data/treering/treering.cfg
  1. Purpose: A simulation of a 9x9 pixel region with a sinusoidal treering dopant variation introduced, so one can see the resulting pixel distortions.
  2. Syntax: src/Poisson data/treering/treering.cfg
  3. Expected run time: $\approx$ 20minutes.
  4. Plot Syntax: python pysrc/VertexPlot.py data/treering/treering.cfg 0 4
- Example 11: data/fe55/pixel.cfg
  1. Purpose: A simulation of a 5x5 pixel region with a single Fe55 event above the center pixel. To determine the distribution of pixel counts, one needs to run many of these simulations with a random center point.
  2. Syntax: src/Poisson data/fe55/pixel.cfg
  3. Expected run time: $\approx$ 5minutes.

4. Plot Syntax: python pysrc/Poisson_Fe55.py data/fe55/pixel.cfg 0 40

- Example 12: data/transrun/trans.cfg

1. Purpose: A simulation of the output transistor of the ITL STA3800C CCD. In order to plot the I-V characteristic, this needs to be re-run with different gate voltage values.
2. Syntax: src/Poisson data/transrun/trans.cfg
3. Expected run time: $\approx$ 10minutes.
4. Plot Syntax: python pysrc/Poisson_Trans.py data/transrun/trans.cfg 0
5. Plot Syntax: python pysrc/ChargePlots_Trans.py data/transrun/trans.cfg 0
6. Plot Syntax: python pysrc/MOSFET_Calculated_IV.py data/transrun/trans.cfg 0

- Example 13: data/satrun/sat.cfg

1. Purpose: A simulation of the saturation level, as determined by the barrier lowering. Again, this needs to be run multiple times with varying parallel voltages.
2. Syntax: src/Poisson data/satrun/sat.cfg
3. Expected run time: $\approx$ 20minutes.
4. Plot Syntax: python pysrc/Poisson_Sat.py data/satrun/sat.cfg 0
5. Plot Syntax: python pysrc/ChargePlots.py data/satrun/sat.cfg 0 9
6. Plot Syntax: python pysrc/Barrier.py data/satrun/sat.cfg 0
7. Plot Syntax: python pysrc/Plot_SatLevel.py data/satrun/sat.cfg 0

- Example 14: data/bfrun/bf.cfg

1. Purpose: This builds up a Gaussian spot in the center of a 9x9 pixel region by sequentially solving Poisson's equation, adding electrons, and repeating. As written, this is done 80 times, building up the central pixel charge to about 80,000 electrons.
2. Syntax: src/Poisson data/bfrun/bf.cfg
3. Expected run time: $\approx$ 8hours.
4. Plot Syntax: python Poisson_Plots.py data/bfrun/bf.cfg 80
5. Plot Syntax: python ChargePlots.py data/bfrun/bf.cfg 80 9
6. Plot Syntax: python Plot_BF_Spots.py data/bfrun/bf.cfg 0 NumSpots 80 (Assumes this was run NumSpots times with random center locations)