PIXELAV2 Operating Instructions

Pixelav2 now comes in 3 different versions (all concurrently supported) that perform 3 different simulation tasks. The version entitled “pixelav2_rndm_trk_n” generates a predefined number of clusters that have the track angles chosen randomly over predefined ranges. The version entitled “pixelav2_list_trkp_n” generates clusters from user supplied track angles (as we do in the CRAFT analyses). And finally, the version entitled “pixelav2_templates_xy” generates a series of single-angle runs that is used to generate templates. All three versions use a pair of common input files and one that is specific to that version. The common inputs are the Bichsel pion-electron cross section file “SIRUTH.SPR”, an input file called “pixel2.init” that contains sensor simulation parameters and also the E-field map for 1/4 of the pixel cell, and a file containing the weighting potential for induction calculations “wgt_pot.init”. The format of the pixel2.init file is as follows:

A single line description of the simulation parameters and field profile

P(GeV/c) Base_Run_Number

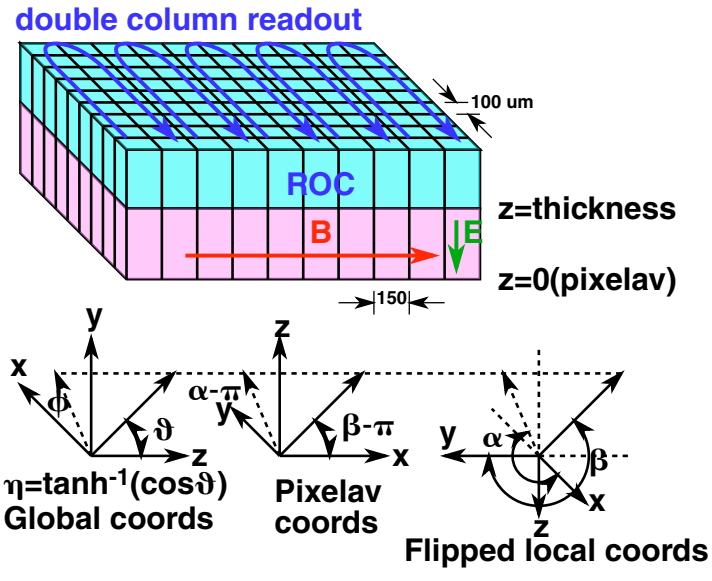
Bx(T) By(T) Bz(T)

thick(μm) sizex(μm) sizey(μm) temp(K) e fluence($10^{14} n_{eq}/cm^2$) h fluence($10^{14} n_{eq}/cm^2$) r_{He} r_{Hh} eh drde nx ny nz

ix iy iz Ex(V/cm) Ey(V/cm) Ez(V/cm) [lots of these to describe the E-field map]

.....

Where: P is the momentum vector of the incident pion: if P<1.1(GeV/c) it is assumed that P=45 GeV/c (their entry points are generated uniformly over the area of one pixel); (Bx,By,Bz) is the magnetic field vector; “thick” is the thickness of the sensor, “sizex” is the x-dimension of the pixel, “sizey” is the y-dimension of the pixel, “temp” is self-explanatory, “e fluence and h fluence” set the trapping rates in irradiated sensors according to the rates given in Kramberger et al, r_{He} and r_{Hh} are the electron and hole Hall factors, eh is an integer index to choose simulation of the n+ side signal (0) or the p+ side signal (1), drde is an integer to choose the original delta-ray range energy relation (0) or a newer one based on NIST Estar cross sections (1), and (nx,ny,nz) define the number of mesh nodes in each direction for the E-field map; there are then nx×ny×nz total lines defining the Efield map. The coordinate system is defined below



where the track direction in pixelav coordinate frame is related to the local sensor coordinate frame as follows: $x_{pixelav}=-y_{local}$, $y_{pixelav}=-x_{local}$, $z_{pixelav}=-z_{local}$, $\cot(\alpha) = \Delta y_{pixelav}/\Delta z_{pixelav} = \Delta x_{local}/\Delta z_{local}$, $\cot(\beta) = \Delta x_{pixelav}/\Delta z_{pixelav} = \Delta y_{local}/\Delta z_{local}$

The third input file for the template-producing pixelav_template_xy code is called run.init and contains a single line

random_seed run_offset num_events sizex_i sizex_f deltax sizey_i sizey_f deltay

where: random_seed (integer) is the random number seed, run_offset is an integer that is added to Run_number from the pixel.init file to calculate the run number NNNN of the first angle point, num_events is the total number of clusters per run to generate, sizex_i is the initial cluster x-size in pixels, sizex_f is the the final cluster x-size, deltax is increment in cluster x-size, sizey_i is the initial cluster y-size in pixels, sizey_f is the the final cluster y-size, and deltay is increment in cluster y-size. The last six quantities describe two nested loops in track angles where the y-size is incremented in the inner loop (it changes first). The program produces the output files "seedfile" and a sequence of consecutively numbered "template_events_dNNNN.out" files. Seedfile contains the current state of the random number generator (it is updated after every event) so that the program can be stopped and restarted at any time without losing the sequence. **It should always be deleted before beginning a new run** (with different input parameters): if it exists, the program will use it to initialize the random number generator, otherwise the program will start from the seed in run.init. The run number NNNN is incremented each time a new set of angles starts.

The random-track-producing pixelav_rndm_trk_n code uses an input file of exactly the format that is also called run.init. The codes differ in that the random track generator produces only a single output run and that the track angles randomly sample the entire angle space defined by the cluster size limits. The cluster size increments are meaningless and are ignored.

The pixelav_list_trkp_n code reads a list of $\cot(\alpha)$,$\cot(\beta)$,momentum,flipped from a file called "track_list.txt". The base run number given in pixel2.init is used as a random number seed.

Each template_events_dNNNN.out file contains the generated events corresponding to a single run. Each event takes TYSIZE+1 (currently 14) lines of output: the first line contains the entry point of the track (x_0,y_0,z_0), the direction of the track (cosx,cosy,cosz), and the total number of e-h pairs generated; lines 2-6 are a TYSIZExTXSIZE (currently 13x21) grid of pixel signals. The origin of the coordinate system is at the center of the front face of pixel (TYSIZE/2,TXSIZE/2) in this grid (counting from 0 in each direction). Note that the sum of the signals is 1/Nscale (1/10 currently) of the number of generated e-h pairs (we only transport 1/10 to save time ... the others don't add anything to the spatial distributions).

Note that pixelav always appends new events to template_events_dNNNN.out. This allows the program to be stopped and started without losing any events. **It is very important to delete these files before beginning any new simulations** (or the old and new events will both be present in the files).

Pixelav always reads and writes files in the local directory. If you wish to run multiple instances of the code, they should be run from different directories (otherwise they will read/write the random number seed and append events to the same files causing very unstable results).

Pixelav runs in two modes. Simply executing the program ./pixelav2_version will cause it to run from your shell. If you type ./pixelav2_version f , the program will fork a detached process and will run disconnected from the shell (you can close the shell or logoff and the process will run for the desired time). If you kill the running job, simply restarting it will begin from where you left-off. If you need additional statistics, you can just run it again with no changes. If you wish to run more than one instance of the code (on a dual processor machine) with the same initial conditions, you should make sure to use different random number seeds in the two initialization files (everything else can be the same) located in two different directories.

To begin a new simulation, there are four steps to follow: 1) remove the existing links to the input files (rm pixel2.init, rm run.init/cosmic_muons.txt), 2) link the new input files to pixel.init, run.init, or track_list.txt, 3) remove old files (rm seedfile, rm template_events_dNNNN.out), 4) start the job (./pixelav_version f).

The template generator call “pixelav2_templates_xy_runlist” reads the same pixel2.init initialization file but also reads a file “runlist.init” containing 1 line per output run (up to 500 output runs) of the following format,

run_offset num_events sizex sizey

where the quantities run_offset and num_events have already been defined. The quantities sizex and sizey define the cluster sizes for this one run. The code takes up to 3 input parameters: index of first run (nth run counting from 1), number of runs (defaults to 1 if not specified), f (if present, it will fork the process). The process will produce a seed storage file “seedfileNNNN” named after the first run number and a series of “template_events_dNNNN.out” files as specified. This process is designed to run multiple jobs from the same directory to work on computing clusters.

Code Distribution

The four codes are distributed in the tarball pixelav.tgz which also contains two simple scripts to compile and link them. The scripts, link_ppc and link_intel, work on ppc and intel machines, respectively. All of the auxilliary code needed to run the processes is included in each file so the local machine needs only a c-compiler and a valid mathlib.

Output File Format

To analyze the output files, it is still necessary to add noise to the individual pixels and to simulate the readout chip analog response. There is a simple fortran program called “testbeam_reformat3” that does this and also changes coordinate system to that used by the test beam analysis code. Normally, the file barrel_ten.out is saved with a 5 digit number ABCDE added to its name “barrel_tenABCDE.out”. **Two lines are added to the front of the file: a header describing the simulation details, and a line containing xsize, ysize, and thick.** The first line is copied into the output file and eventually to the root file. The file number and other information is passed to testbeam_reformat3 by an input file called “pixel.proc”. The file contains a single ascii line:

ABCD .true. noise(float) threshold(float)

where the second and fourth entries are not used by testbeam_reformat3. The third entry noise is the average noise in electrons (I use 500. as a typical value). The program writes an output file called “tbsimulationABCD.out”. We have a root script that converts this to the same root format used in the testbeam analysis.

Automatic writing of the barrel_ten.out header lines can be switched-on by starting the pixel.init header (first line of that file) with the character ‘1’ [instead of beginning “Sintef sensor”, begin “1Sintef sensor”]. Output of the character switch ‘1’ to barrel_ten.out is suppressed. Do not use this option if you intend to combine several output files from several runs begun with different random number seeds (or you could inadvertently insert a header into the middle of a file). Note that the header is not written if a seedfile is found (you can safely stop and restart runs).